gatk

The GATK Guide Book

Version 2.7-4

BROAD INSTITUTE

C. Broad Institute 2012

About this Guide Book

This Guide Book is a collection of all the documentation articles that supplement the Technical Documentation (which can be generated from the source code of the program). We provide this as a PDF file mainly to serve as a versioned record of the supplemental documentation. Of course, it can also be conveniently used for offline reading and for printing, although we ask you to avoid printing the entire volume in the interest of preserving the planet's trees!

The articles contained herein are grouped in 8 main sections as listed below:

- Introductory Materials
- Best Practices
- Methods and Workflows
- FAQs
- Tutorials
- Developer Zone
- Third-Party Tools
- Version History

You can find a complete list of article titles and their corresponding page numbers indexed in the Table Of Contents, which is located at the end of this volume.

Introductory Materials

If you are new to the GATK, the following articles will give you an overview of what it is and what it can do. At the end of this section, you will find a list of links to more in-depth articles on introductory topics to get you started in practice.

What is the GATK?

Simply what it says on the can: a Toolkit for Genome Analysis

Say you have ten exomes and you want to identify the rare mutations they all have in common -- the GATK can do that. Or you need to know which mutations are specific to a group of patients, as opposed to a healthy cohort -- the GATK can do that too. In fact, the GATK is the industry standard for such analyses.

But wait, there's more!

Because of the way it is built, the GATK is highly generic and can be applied to all kinds of datasets and genome analysis problems. It can be used for discovery as well as for validation. It's just as happy handling exomes as whole genomes. It can use data generated with a variety of different sequencing technologies. And although it was originally developed for human genetics, the GATK has evolved to handle genome data from any organism, with any level of ploidy. Your plant has six copies of each chromosome? Bring it on.

So what's in the can?

At the heart of the GATK is an industrial-strength infrastructure and engine that handle data access, conversion and traversal, as well as high-performance computing features. On top of that lives a rich ecosystem of specialized tools, called **walkers**, that you can use out of the box, individually or chained into scripted workflows, to perform anything from simple data diagnostics to complex **reads-to-results** analyses.

Please see the Technical Documentation section for a complete list of tools and their capabilities.

Using the GATK
Get started today

Platform and requirements

The GATK is designed to run on Linux and other POSIX-compatible platforms. Yes, that includes MacOS X! If you are on any of the above, see the Downloads section for downloading and installation instructions. Note that you will need to have Java installed to run the GATK, and some tools additionally require R to generate PDF plots. If you're stuck with Windows, you're not completely out of luck -- it's possible to use the GATK with Cygwin, although we can't provide any specific support for that. If you're on something else... no, there are no plans to port the GATK to Android or iOS in the near future.

Interface

Now here's the kicker: the GATK does not have a graphical user interface. All tools are called via the command-line interface. If that is not something you are used to, or you have no idea what that even means, don't worry. It's easier to learn than you might think, and there are many good online tutorials that can get help you get comfortable with the command-line environment. Before you know it you'll be writing scripts to chain tools together into workflows... You don't need to have any programming experience to use the GATK, but you might pick some up along the way!

Command structure and tool arguments

All the GATK tools are called using the same basic command structure. Here's a simple example that counts the number of sequence reads in a BAM file:

```
java -jar GenomeAnalysisTK.jar \
     -T CountReads \
     -R example_reference.fasta \
     -I example_reads.bam
```

The `-jar` argument invokes the GATK engine itself, and the `-T` argument tells it which tool you want to run. Arguments like `-R` for the genome reference and `-I` for the input file are also given to the GATK engine and can be used with all the tools (see complete list of available arguments for the GATK engine. Most tools also take additional arguments that are specific to their function. These are listed for each tool on that tool's documentation page, all easily accessible through the Technical Documentation index.

High Performance
Built for scalability and parallelism

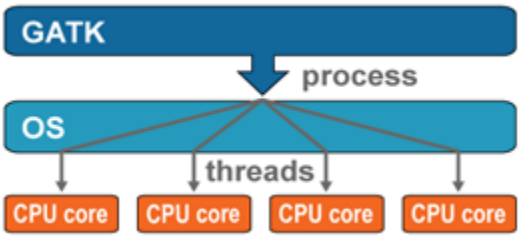
The GATK was built from the ground up with performance in mind.

Map/Reduce: it's not just for Google anymore

Every GATK walker is built using the Map/Reduce framework, which is basically a strategy to speed up performance by breaking down large iterative tasks into shorter segements then merging overall results.

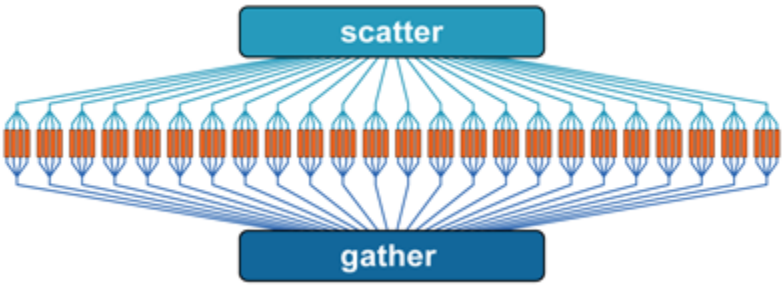
Multi-threading

The GATK takes advantage of the latest processors using multi-threading, *i. e.* run using multiple cores on the same machine, sharing the RAM. To enable multi-threading in the GATK, simply add the `-nt x` and/or `-nct x` arguments to your command line, where `x` is the number of threads or cores you want to use. See the documentation on parallelism for more details on these arguments' capabilities.



Out on the farm with Queue

Queue is a companion program that allows the GATK to take parallelization to the next level: running jobs on a high-performance computing cluster, or server farm. Queue manages the entire process of breaking down big jobs into many smaller ones (scatter) then collecting and merging results when they are done (gather). At the Broad, we use a Queue pipeline to run GATK analyses on hundreds, even thousands of exomes, on our cluster of hundreds of nodes.



Queue uses a scatter-gather process to parallelize operations.

Which GATK package is right for you?

GATK Framework | Broad GATK | Appistry GATK

There are three distinct GATK packages available:

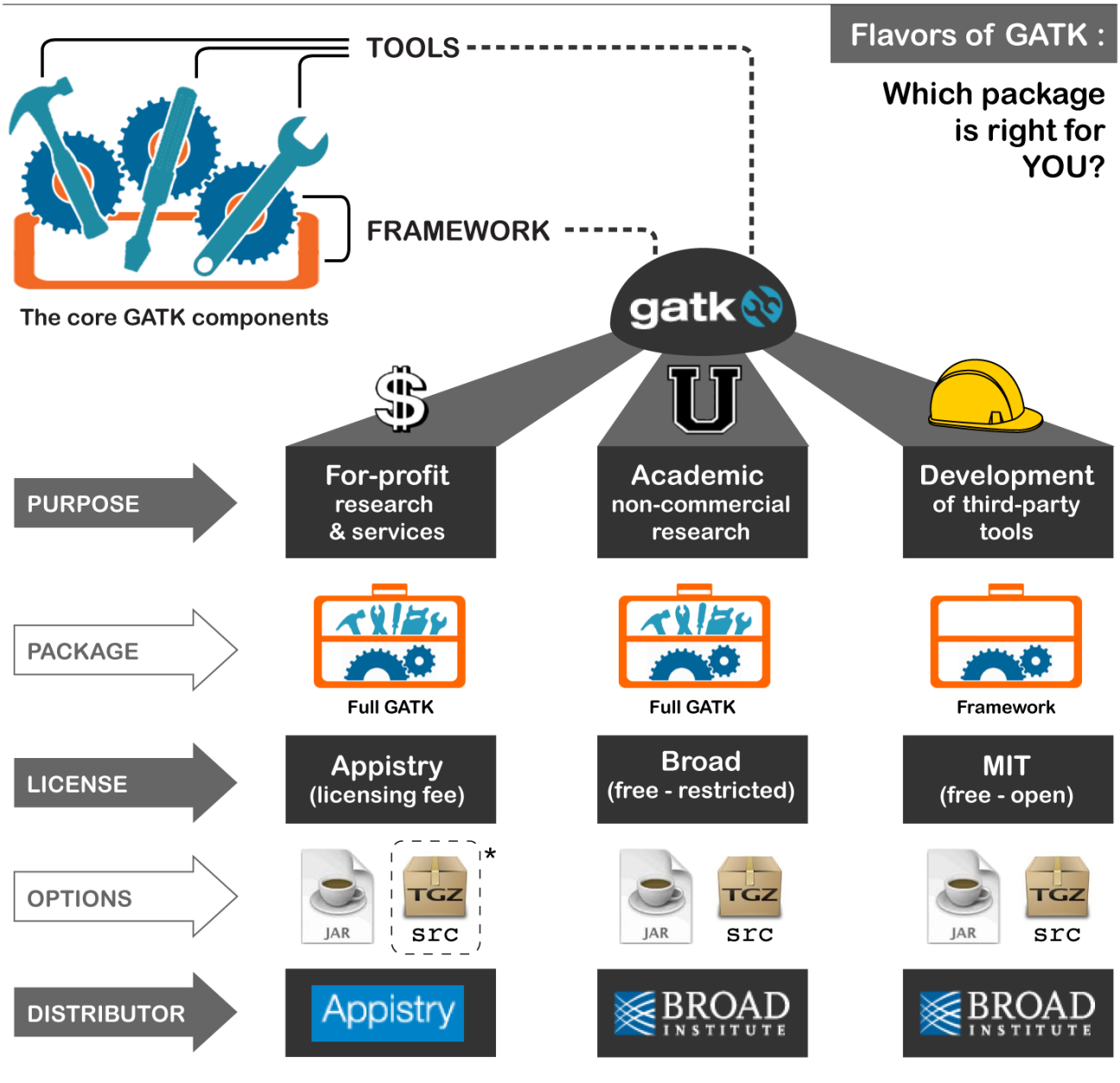
- The **GATK Framework** package contains the GATK engine, core libraries and utility tools. It is a programming framework meant for developers who build their own third-party tools on top of the GATK engine. It is released under the MIT license and the source code is freely available to all on our Github repository.

- The **Broad GATK** package contains the full GATK suite of tools. It is released under a Broad Institute license that restricts its use to non-commercial activities. It is available free of charge to academic and non-profit researchers who use it for those purposes. A precompiled binary of the program (.jar file) is

available for download from our website, and the source code is available on our Github repository.

- The **Appistry GATK** package contains the full GATK suite of tools licensed for commercial use by our partner, Appistry. Please contact Appistry to purchase a license and obtain the program. Licensed users through Appistry, in addition to having access to the full GATK and the added benefits of a fully-fledged commercial solution (less buggy, more help-y), may optionally purchase access to the source code.

The following figure summarizes the different packages and their corresponding licenses.



* Because the GATK from Appistry is a supported product, Appistry has restricted access to the source code depending upon the license. Please contact Appistry for details.

Introductory Materials

List of articles for beginners

These are the articles you should start out with if you're new to the GATK. You can look them up in this Guide Book by category (based on the icon) or on our website by article number.

- (howto) Run the GATK for the first time (#1209)
- (howto) Test your Queue installation (#1287)
- A primer on parallelism with the GATK (#1988)
- Best Practice Variant Detection with the GATK v4, for release 2.0 [RETIRED] (#1186)
- How can I prepare a FASTA file to use as reference? (#1601)
- How should I interpret VCF files produced by the GATK? (#1268)
- Overview of Queue (#1306)
- What are the prerequisites for running GATK? (#1852)
- What input files does the GATK accept? (#1204)
- What is "Phone Home" and how does it affect me? (#1250)
- What is GATK-Lite and how does it relate to "full" GATK 2.x? [RETIRED] (#1720)
- What is Map/Reduce and why are GATK tools called "walkers"? (#1754)
- What's in the resource bundle and how can I get it? (#1213)

Best Practices

This reads-to-results variant calling workflow lays out the best practices recommended by our group for all the steps involved in calling variants with the GATK. It is used in production at the Broad Institute on every genome that rolls out of the sequencing facility. In addition to the recommendations detailed in the following pages, you can also find relevant presentation slides and videos on the Events page of our website.

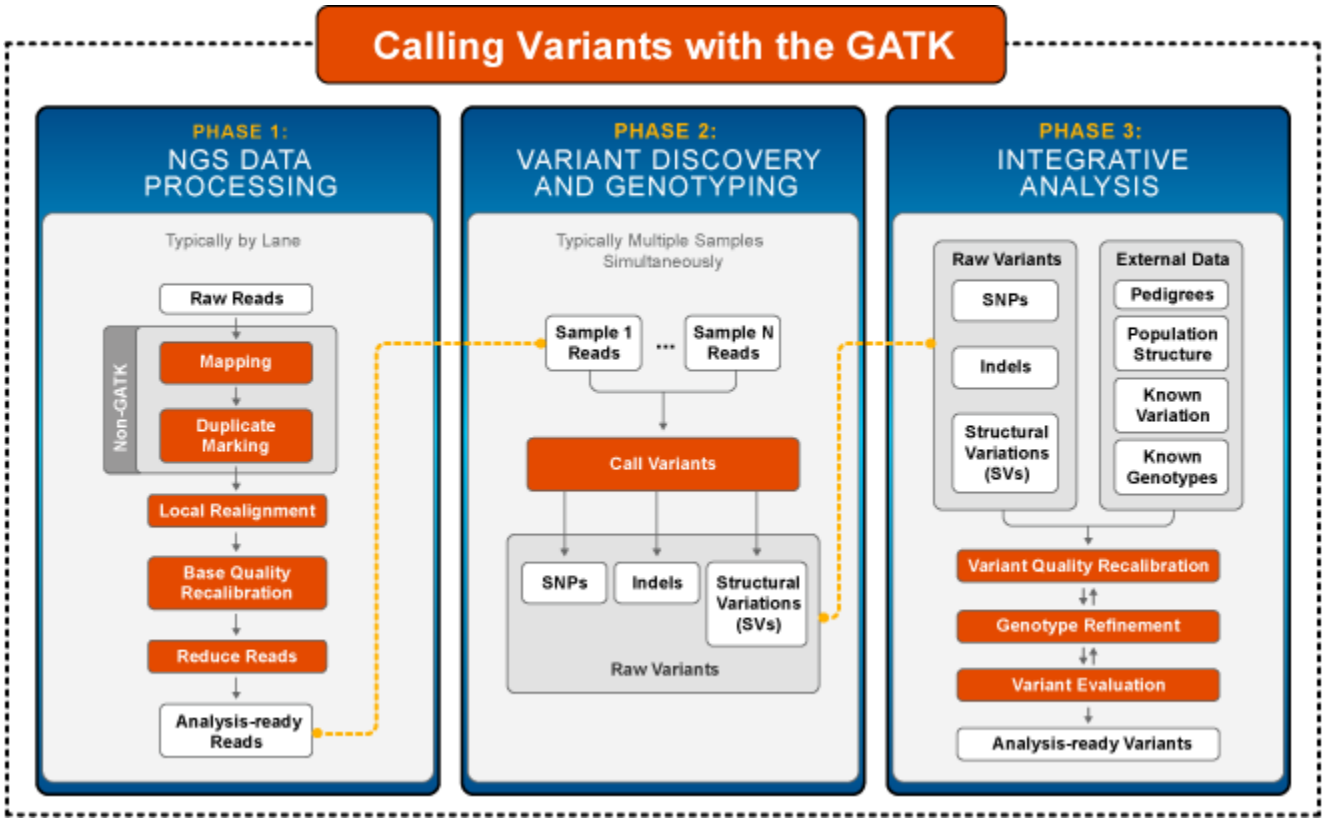
Best Practice Variant Detection with the GATK v4, for release 2.0 [RETIRED] #1186

Last updated on 2013-09-13 21:12:44

Introduction

1. The basic workflow

Our current best practice for making SNP and indel calls is divided into four sequential steps: initial mapping, refinement of the initial reads, multi-sample indel and SNP calling, and finally variant quality score recalibration. These steps are the same for targeted resequencing, whole exomes, deep whole genomes, and low-pass whole genomes.



Example commands for each tool are available on the individual tool's wiki entry. There is also a list of which resource files to use with which tool.

Note that due to the specific attributes of a project the specific values used in each of the commands may need to be selected/modified by the analyst. Care should be taken by the analyst running our tools to understand what each parameter does and to evaluate which value best fits the data and project design.

2. Lane, Library, Sample, Cohort

There are four major organizational units for next-generation DNA sequencing processes that used throughout this documentation:

- Lane: The basic machine unit for sequencing. The lane reflects the basic independent run of an NGS machine. For Illumina machines, this is the physical sequencing lane.
- Library: A unit of DNA preparation that at some point is physically pooled together. Multiple lanes can be run from aliquots from the same library. The DNA library and its preparation is the natural unit that is being sequenced. For example, if the library has limited complexity, then many sequences are duplicated and will result in a high duplication rate across lanes.
- Sample: A single individual, such as human CEPH NA12878. Multiple libraries with different properties can be constructed from the original sample DNA source. Here we treat samples as independent individuals whose genome sequence we are attempting to determine. From this perspective, tumor / normal samples are different despite coming from the same individual.
- Cohort: A collection of samples being analyzed together. This organizational unit is the most subjective and depends intimately on the design goals of the sequencing project. For population discovery projects like the 1000 Genomes, the analysis cohort is the ~100 individual in each population. For exome projects with many samples (e.g., ESP with 800 EOMI samples) deeply sequenced we divide up the complete set of samples into cohorts of ~50 individuals for multi-sample analyses.

This document describes how to call variation within a single analysis cohort, comprised for one or many samples, each of one or many libraries that were sequenced on at least one lane of an NGS machine.

Note that many GATK commands can be run at the lane level, but will give better results seeing all of the data for a single sample, or even all of the data for all samples. Unfortunately, there's a trade-off in computational cost by running these commands across all of your data simultaneously.

3. Testing data: 64x HiSeq on chr20 for NA12878

In order to help individuals get up to speed, evaluate their command lines, and generally become familiar with the GATK tools we recommend you download the raw and realigned, recalibrated NA12878 test data from the GATK resource bundle. It should be possible to apply all of the approaches outlined below to get excellent results for realignment, recalibration, SNP calling, indel calling, filtering, and variant quality score recalibration using this data.

4. Where can I find out more about the new GATK 2.0 tools you are talking about?

In our GATK 2.0 slide archive.

Phase I: Raw data processing

1. Raw FASTQs to raw reads via mapping

The GATK data processing pipeline assumes that one of the many NGS read aligners (see this review) has been applied to your raw FASTQ files. For Illumina data we recommend BWA because it is accurate, fast, well-supported, open-source, and emits SAM files natively (which are then easy to convert to BAM).

2. Raw reads to analysis-ready reads

The three key processes used here are:

- Local realignment around indels: Reads that align on the edges of indels often get mapped with mismatching bases that might look like evidence for SNPs. We look for the most consistent placement of the reads with respect to the indel in order to clean up these artifacts.

- MarkDuplicates: Duplicately sequenced molecules shouldn't be counted as additional evidence for or against a putative variant. By marking these reads as duplicates the algorithms in the GATK know to ignore them.

- Base quality score recalibration: The per-base estimate of error known as the base quality score is the foundation upon which all statistically calling algorithms are based. We've found that the estimates provided by the sequencing machines are often inaccurate, and worse, biased. Through recalibration an empirically accurate error model is assigned to the bases to create an analysis-ready bam file. Note: if you have old data that has been recalibrated with an old version of BQSR, you need to rerun your data with the new version so insertion and deletion qualities can be added to your recalibrated BAM file.

There are several options here from the easy and fast basic protocol to the more comprehensive but computationally expensive pipeline. For example, there are two types of realignment which constitute a vastly different amount of processing power required:

- Realignment only at known sites, which is very efficient, can operate with little coverage (1x per lane genome wide) but can only realign reads at known indels.

- Fully local realignment uses mismatching bases to determine if a site should be realigned, and relies on sufficient coverage to discover the correct indel allele in the reads for alignment. It is much slower (involves SW step) but can discover new indel sites in the reads. If you have a database of known indels (for human, this database is extensive) then at this stage you would also include these indels during realignment, which vastly improves sensitivity, specificity, and speed.

Fast: lane-level realignment (at known sites only) and lane-level recalibration

This protocol uses lane-level local realignment around known indels (very fast, as there's no sample level processing) to clean up lane-level alignments. This results in better quality scores, as they are less biased for indel alignment artefacts.

```
for each lane.bam
    dedup.bam <- MarkDuplicate(lane.bam)
    realigned.bam <- realign(dedup.bam) [at only known sites, if possible, otherwise skip]
    recal.bam <- recal(realigned.bam)
```

Fast + per-sample processing

Here we are essentially just merging the recalibrated lane.bams for a sample, dedupping the reads, and calling it quite. It doesn't perform indel realignment across lanes, so it leaves in some indels artifacts. For humans, which now have an extensive list of indels (get them from the GATK bundle!) the lane-level realignment around known indels is going to make up for the lack of cross-lane realignment. This protocol is appropriate if you are going to use callers like the HaplotypeCaller, UnifiedGenotyper with BAQ, or samtools with BAQ that are less sensitive to the initial alignment of reads, or if your project has limited coverage per sample (< 8x) where per-sample indel realignment isn't more empowered than per-lane realignment. For other situations or for organisms with limited database of segregating indels, it's better to use the advanced protocol if you have deep enough data per sample.

```
for each sample
    recals.bam <- merged lane-level recal.bams for sample
    dedup.bam <- MarkDuplicates(recals.bam)
    sample.bam <- dedup.bam
```

Better: recalibration per lane then per-sample realignment with known indels

As with the basic protocol, this protocol assumes the per-lane processing has been already completed. This protocol is essentially the basic protocol but with per-sample indel realignment.

```
for each sample
    recals.bam <- merged lane-level recal.bams for sample
    dedup.bam <- MarkDuplicates(recals.bam)
    realigned.bam <- realign(dedup.bam) [with known sites included if available]
    sample.bam <- realigned.bam
```

This is the protocol we use at the Broad in our fully automated pipeline because it gives an optimal balance of performance, accuracy and convenience.

Best: per-sample realignment with known indels then recalibration

Rather than doing the lane level cleaning and recalibration, this process aggregates all of the reads for each

sample and then does a full deduppping, realign, and recalibration, yielding the best single-sample results. The big change here is sample-level cleaning followed by recalibration, giving you the most accurate quality scores possible for a single sample.

```
for each sample
    lanes.bam <- merged lane.bams for sample
    dedup.bam <- MarkDuplicates(lanes.bam)
    realigned.bam <- realign(dedup.bam) [with known sites included if available]
    recal.bam <- recal(realigned.bam)
    sample.bam <- recal.bam
```

This protocol can be hard to implement in practice unless you can afford to wait until all of the data is available to do data processing for your samples.

Misc. notes on the process

- MarkDuplicates needs only be run at the library level. So the sample-level deduppping isn't necessary if you only ever have a library on a single lane. If you run the sample library on many lanes (as can be necessary for whole exome, for example), you should dedup at the library level.

- The base quality score recalibrator is read group aware, so running it on a merged BAM files containing multiple read groups is the same as running it on each bam file individually. There's some memory cost (so it's best not to recalibrate many read groups simultaneously) but for reasonable projects this is fine.

- Local realignment preserves read meta-data, so you can realign and then recalibrate just fine.

- Multi-sample realignment with known sites and recalibration isn't really recommended any longer. It's extremely computational expensive and isn't necessary for advanced callers with advanced filters like the Unified Genotyper / HaplotypeCaller and VQSR. It's better to use one of the protocols above and then an advanced caller that is robust to indel artifacts.

- However, note that for contrastive calling projects -- such as cancer tumor/normals -- we recommend realigning both the tumor and the normal together in general to avoid slight alignment differences between the two tissue types.

3. Reducing BAMs to minimize file sizes and improve calling performance

ReduceReads is a novel (perhaps even breakthrough?) GATK 2.0 data compression algorithm. The purpose of ReducedReads is to take a BAM file with NGS data and reduce it down to just the information necessary to make accurate SNP and indel calls, as well as genotype reference sites (hard to achieve) using GATK tools like UnifiedGenotyper or HaplotypeCaller. ReduceReads accepts as an input a BAM file and produces a valid BAM file (it works in IGV!) but with a few extra tags that the GATK can use to make accurate calls.

You can find more information about reduced reads in some of our presentations in the archive.

ReduceReads works well for exomes or high-coverage (at least 20x average coverage) whole genome BAM files. In this case we highly recommend using ReduceReads to minimize the file sizes. Note that ReduceReads performs a lossy compression of the sequencing data that works well with the downstream GATK tools, but may

not be supported by external tools. Also, we recommend that you archive your original BAM file, or at least a copy of your original FASTQs, as ReduceReads is highly lossy and doesn't quality as an archive data compression format.

Using ReduceReads on your BAM files will cut down the sizes to approximately 1/100 of their original sizes, allowing the GATK to process tens of thousands of samples simultaneously without excessive IO and processing burdens. Even for single samples ReduceReads cuts the memory requirements, IO burden, and CPU costs of downstream tools significantly (10x or more) and so we recommend you preprocess analysis-ready BAM files with ReducedReads.

```
for each sample
    sample.reduced.bam <- ReduceReads(sample.bam)
```

Phase II: Initial variant discovery and genotyping

1. Input BAMs for variant discovery and genotyping

After the raw data processing step, the GATK variant detection process assumes that you have aligned, duplicate marked, and recalibrated BAM files for all of the samples in your cohort. Because the GATK can dynamically merge BAM files, it isn't critical to have merged files by lane into sample bams, or even samples bams into cohort bams. In general we try to create sample level bams for deep data sets (deep WG or exomes) and merged cohort files by chromosome for WG low-pass.

For this part of the this document, I'm going to assume that you have a single realigned, recalibrated, dedupped BAM per sample, called sampleX.bam, for X from 1 to N samples in your cohort. Note that some of the data processing steps, such as multiple sample local realignment, will merge BAMS for many samples into a single BAM. If you've gone down this route, you just need to modify the GATK commands as necessary to take not multiple BAMs, one for each sample, but a single BAM for all samples.

2. Multi-sample SNP and indel calling

The next step in the standard GATK data processing pipeline, whole genome or targeted, deep or shallow, is to apply the Haplotype Caller or Unified Genotyper to identify sites among the cohort samples that are statistically non-reference. This will produce a multi-sample VCF file, with sites discovered across samples and genotypes assigned to each sample in the cohort. It's in this stage that we use the meta-data in the BAM files extensively -- read groups for reads, with samples, platforms, etc -- to enable us to do the multi-sample merging and genotyping correctly. It was a pain for data processing, yes, but now life is easy for downstream calling and analysis.

Selecting an appropriate quality score threshold

A common question is the confidence score threshold to use for variant detection. We recommend:

- Deep (> 10x coverage per sample) data: we recommend a minimum confidence score threshold of Q30.

- Shallow (< 10x coverage per sample) data: because variants have by necessity lower quality with shallower coverage we recommend a minimum confidence score of Q4 in projects with 100 samples or fewer and Q10 otherwise.

Standard protocol: HaplotypeCaller

```
raw.vcf <- HaplotypeCaller(sample1.bam, sample2.bam, ..., sampleN.bam)
```

Standard protocol: UnifiedGenotyper

```
raw.vcf <- UnifiedGenotyper(sample1.bam, sample2.bam, ..., sampleN.bam)
```

Choosing HaplotypeCaller or UnifiedGenotyper

- We believe the best possible caller in the GATK is the HaplotypeCaller, which combines a local de novo assembler with a more advanced HMM likelihood function than the UnifiedGenotyper. It should produce excellent SNP, MNP, indel, and short SV calls. It should be the go-to calling algorithm for most projects. It is, for example, how we make our Phase II call set for 1000 Genomes.

- Currently the HaplotypeCaller only supports diploid calling. If you want to call non-diploid samples you'll need to use the UnifiedGenotyper.

- At the moment the HaplotypeCaller does not support multithreading. For now you should indeed stick with the UG if you wish to use the `-nt` option. However you can use Queue to parallelize execution of HaplotypeCaller.

- If for some reason you cannot use the HaplotyperCaller do fall back to the UnifiedGenotyper protocol below. Otherwise try out the HaplotypeCaller and let us know about your experiences here on the forum!

Phase III: Integrating analyses: getting the best call set possible

This raw VCF file should be as sensitive to variation as you'll get without imputation. At this stage, you can assess things like sensitivity to known variant sites or genotype chip concordance. The problem is that the raw VCF will have many sites that aren't really genetic variants but are machine artifacts that make the site statistically non-reference. All of the subsequent steps are designed to separate out the false positive machine artifacts from the true positive genetic variants.

1. Statistical filtering of the raw calls

The process used here is the Variant quality score recalibrator which builds an adaptive error model using known variant sites and then applies this model to estimate the probability that each variant in the callset is a true genetic variant or a machine/alignment artifact. All filtering criteria are learned from the data itself.

2. Analysis ready VCF protocol

Take a look at our FAQ page for specific recommendations on which training sets and command-line arguments to use with various project designs. It is expected that analysis will be required by the user when determining the best way to use this tool for different projects. These recommendations are only meant as jumping off points!

We've found that the best results are obtained with the VQSR when it is used to build separate adaptive error models for SNPs versus INDELs:

```
    snp.model <- BuildErrorModelWithVQSR(raw.vcf, SNP)
    indel.model <- BuildErrorModelWithVQSR(raw.vcf, INDEL)
    recalibratedSNPs.rawIndels.vcf <- ApplyRecalibration(raw.vcf, snp.model, SNP)
    analysisReady.vcf <- ApplyRecalibration(recalibratedSNPs.rawIndels.vcf, indel.model,
INDEL)
```

3. Notes about small whole exome projects or small target experiments

In our testing we've found that in order to achieve the best exome results one needs to use an exome callset with at least 30 samples. Also, for experiments that employ targeted resequencing of a small region (for example, a few hundred genes), VQSR may not be empowered regardless of the number of samples in the experiment. For users with experiments containing fewer exome samples or with a small target region there are several options to explore (listed in priority order of what we think will give the best results):

- Add additional samples for variant calling, either by sequencing additional samples or using publicly available exome bams from the 1000 Genomes Project (this option is used by the Broad exome production pipeline).

- Use the VQSR with the smaller SNP callset but experiment with the argument settings. For example, try adding `--maxGaussians 4 --percentBad 0.05` to your command line. Note that this is very dependent on your dataset, and you may need to try some very different settings. It may even not work at all. Unfortunately we cannot give you any specific advice, so please do not post questions on the forum asking for help finding the right parameters.

- Use hard filters (detailed below).

Recommendations for very small data sets (in terms of both number of samples or size of targeted regions)

These recommended arguments for VariantFiltration are only to used when ALL other options are not available:

You will need to compose filter expressions (see here, here and here for details) to filter on the following annotations and values:

For SNPs:

- QD < 2.0
- MQ < 40.0
- FS > 60.0
- HaplotypeScore > 13.0
- MQRankSum < -12.5
- ReadPosRankSum < -8.0

For indels:

- QD < 2.0
- ReadPosRankSum < -20.0
- InbreedingCoeff < -0.8
- FS > 200.0

Note that the InbreedingCoeff statistic is a population-level calculation that is only available with 10 or more samples. If you have fewer samples you will need to omit that particular filter statement.

For shallow-coverage (<10x): you cannot use filtering to reliably separate true positives from false positives. You must use the protocol involving variant quality score recalibration.

The maximum DP (depth) filter only applies to whole genome data, where the probability of a site having exactly N reads given an average coverage of M is a well-behaved function. First principles suggest this should be a binomial sampling but in practice it is more a Gaussian distribution. Regardless, the DP threshold should be set a 5 or 6 sigma from the mean coverage across all samples, so that the DP > X threshold eliminates sites with excessive coverage caused by alignment artifacts. Note that **for exomes, a straight DP filter shouldn't be used** because the relationship between misalignments and depth isn't clear for capture data.

That said, all of the caveats about determining the right parameters, etc, are annoying and are largely eliminated by variant quality score recalibration.

Methods and Workflows

The documentation articles in this section cover:

- **Methods using individual tools:** articles providing recommendations on how to apply the tools on your data to answer specific questions or achieve certain data transformations. These articles are meant to complement the Technical Documentation available for each tool.
- **Workflows using several tools:** articles describing how to chain several tools together appropriately into multi-step analyses and pipelines.
- **Computational methods:** these articles describe how the GATK tools work and how to use them efficiently.

Please note that while many of the articles contain command lines and argument values, these are given as examples only and may not be the most appropriate for your dataset. It is your responsibility to ascertain that the parameters you use for analysis make sense considering your experimental design and materials.

In addition, certain examples, argument names, usages and values may become obsolete over time. We try to update the documentation regularly but some articles may fall through the net. This occasionally leads to apparent contradictions between articles in this section and the Technical Documentation that is available for each tool. When in doubt, keep in mind that the Technical Documentation is updated more frequently and always trumps other documentation sources.

A primer on parallelism with the GATK #1988

Last updated on 2013-01-26 05:10:36

This document explains the concepts involved and how they are applied within the GATK (and Queue where applicable). For specific configuration recommendations, see the companion document on parallelizing GATK tools.

1. Introducing the concept of parallelism

Parallelism is a way to make a program finish faster by performing several operations in parallel, rather than sequentially (i.e. waiting for each operation to finish before starting the next one).

Imagine you need to cook rice for sixty-four people, but your rice cooker can only make enough rice for four people at a time. If you have to cook all the batches of rice sequentially, it's going to take all night. But if you have eight rice cookers that you can use in parallel, you can finish up to eight times faster.

This is a very simple idea but it has a key requirement: you have to be able to break down the job into smaller tasks that can be done independently. It's easy enough to divide portions of rice because rice itself is a collection of discrete units. In contrast, let's look at a case where you can't make that kind of division: it takes one pregnant woman nine months to grow a baby, but you can't do it in one month by having nine women share the work.

The good news is that most GATK runs are more like rice than like babies. Because GATK tools are built to use the Map/Reduce method (see doc for details), most GATK runs essentially consist of a series of many small independent operations that can be parallelized.

A quick warning about tradeoffs

Parallelism is a great way to speed up processing on large amounts of data, but it has "overhead" costs. Without getting too technical at this point, let's just say that parallelized jobs need to be managed, you have to set aside memory for them, regulate file access, collect results and so on. So it's important to balance the costs against the benefits, and avoid dividing the overall work into too many small jobs.

Going back to the introductory example, you wouldn't want to use a million tiny rice cookers that each boil a single grain of rice. They would take way too much space on your countertop, and the time it would take to distribute each grain then collect it when it's cooked would negate any benefits from parallelizing in the first place.

Parallel computing in practice (sort of)

OK, parallelism sounds great (despite the tradeoffs caveat), but how do we get from cooking rice to executing programs? What actually happens in the computer?

Consider that when you run a program like the GATK, you're just telling the computer to execute a set of instructions.

Let's say we have a text file and we want to count the number of lines in it. The set of instructions to do this can be as simple as:

```
- open the file, count the number of lines in the file, tell us the number, close the file
```

Note that `tell us the number` can mean writing it to the console, or storing it somewhere for use later on.

Now let's say we want to know the number of words on each line. The set of instructions would be:

```
- open the file, read the first line, count the number of words, tell us the number, read the second line, count the number of words, tell us the number, read the third line, count the number of words, tell us the number
```

And so on until we've read all the lines, and finally we can close the file. It's pretty straightforward, but if our file has a lot of lines, it will take a long time, and it will probably not use all the computing power we have available.

So to parallelize this program and save time, we just cut up this set of instructions into separate subsets like this:

```
- open the file, index the lines
- read the first line, count the number of words, tell us the number
- read the second line, count the number of words, tell us the number
- read the third line, count the number of words, tell us the number
```

- `[repeat for all lines]`
- `collect final results and close the file`

Here, the `read the Nth line` steps can be performed in parallel, because they are all independent operations.

You'll notice that we added a step, `index the lines`. That's a little bit of peliminary work that allows us to perform the `read the Nth line` steps in parallel (or in any order we want) because it tells us how many lines there are and where to find each one within the file. It makes the whole process much more efficient. As you may know, the GATK requires index files for the main data files (reference, BAMs and VCFs); the reason is essentially to have that indexing step already done.

Anyway, that's the general principle: you transform your linear set of instructions into several subsets of instructions. There's usually one subset that has to be run first and one that has to be run last, but all the subsets in the middle can be run at the same time (in parallel) or in whatever order you want.

2. Parallelizing the GATK

There are three different modes of parallelism offered by the GATK, and to really understand the difference you first need to understand what are the different levels of computing that are involved.

A quick word about levels of computing

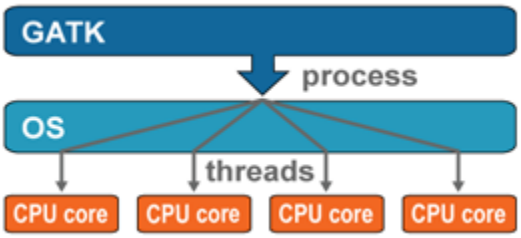
By levels of computing, we mean the computing units in terms of hardware: the core, the machine (or CPU) and the cluster.

- **Core:** the level below the machine. On your laptop or desktop, the CPU (central processing unit, or processor) contains one or more cores. If you have a recent machine, your CPU probably has at least two cores, and is therefore called dual-core. If it has four, it's a quad-core, and so on. High-end consumer machines like the latest Mac Pro have up to twelve-core CPUs (which should be called dodeca-core if we follow the Latin terminology) but the CPUs on some professional-grade machines can have tens or hundreds of cores.
- **Machine:** the middle of the scale. For most of us, the machine is the laptop or desktop computer. Really we should refer to the CPU specifically, since that's the relevant part that does the processing, but the most common usage is to say **machine**. Except if the machine is part of a cluster, in which case it's called a **node**.
- **Cluster:** the level above the machine. This is a high-performance computing structure made of a bunch of machines (usually called **nodes**) networked together. If you have access to a cluster, chances are it either belongs to your institution, or your company is renting time on it. A cluster can also be called a **server farm** or a **load-sharing facility**.

Parallelism can be applied at all three of these levels, but in different ways of course, and under different names. Parallelism takes the name of **multi-threading** at the core and machine levels, and **scatter-gather** at the cluster level.

Multi-threading

In computing, a **thread of execution** is a set of instructions that the program issues to the processor to get work done. In **single-threading mode**, a program only sends a single thread at a time to the processor and waits for it to be finished before sending another one. In **multi-threading mode**, the program may send several threads to the processor at the same time.



Not making sense? Let's go back to our earlier example, in which we wanted to count the number of words in each line of our text document. Hopefully it is clear that the first version of our little program (one long set of sequential instructions) is what you would run in single-threaded mode. And the second version (several subsets of instructions) is what you would run in multi-threaded mode, with each subset forming a separate thread. You would send out the first thread, which performs the preliminary work; then once it's done you would send the "middle" threads, which can be run in parallel; then finally once they're all done you would send out the final thread to clean up and collect final results.

If you're still having a hard time visualizing what the different threads are like, just imagine that you're doing cross-stitching. If you're a regular human, you're working with just one hand. You're pulling a needle and thread (a single thread!) through the canvas, making one stitch after another, one row after another. Now try to imagine an octopus doing cross-stitching. He can make several rows of stitches at the same time using a different needle and thread for each. Multi-threading in computers is surprisingly similar to that.

Hey, if you have a better example, let us know in the forum and we'll use that instead.

Alright, now that you understand the idea of multithreading, let's get practical: how do we do get the GATK to use multi-threading?

There are two options for multi-threading with the GATK, controlled by the arguments `-nt` and `-nct`, respectively. They can be combined, since they act at different levels of computing:

- `-nt` / `--num_threads` controls the number of **data threads** sent to the processor (acting at the **machine** level)
- `-nct` / `--num_cpu_threads_per_data_thread` controls the number of **CPU threads** allocated to each data thread (acting at the **core** level).

Not all GATK tools can use these options due to the nature of the analyses that they perform and how they traverse the data. Even in the case of tools that are used sequentially to perform a multi-step process, the individual tools may not support the same options. For example, at time of writing (Dec. 2012), of the tools

involved in local realignment around indels, RealignerTargetCreator supports `-nt` but not `-nct`, while IndelRealigner does not support either of these options.

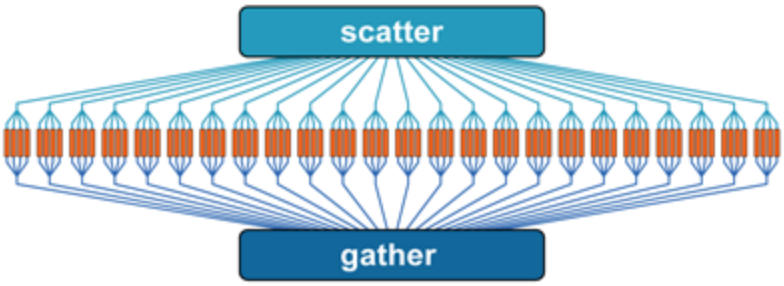
In addition, there are some important technical details that affect how these options can be used with optimal results. Those are explained along with specific recommendations for the main GATK tools in a companion document on parallelizing the GATK.

Scatter-gather

If you Google it, you'll find that the term **scatter-gather** can refer to a lot of different things, including strategies to get the best price quotes from online vendors, methods to control memory allocation andâ€¦ an indie-rock band. What all of those things have in common (except possibly the band) is that they involve breaking up a task into smaller, parallelized tasks (scattering) then collecting and integrating the results (gathering). That should sound really familiar to you by now, since it's the general principle of parallel computing.

So yes, "scatter-gather" is really just another way to say we're parallelizing things. OK, but how is it different from multithreading, and why do we need yet another name?

As you know by now, multithreading specifically refers to what happens internally when the program (in our case, the GATK) sends several sets of instructions to the processor to achieve the instructions that you originally gave it in a single command-line. In contrast, the scatter-gather strategy as used by the GATK involves a separate program, called Queue, which generates separate GATK jobs (each with its own command-line) to achieve the instructions given in a so-called Qscript (i.e. a script written for Queue in a programming language called Scala).



At the simplest level, the Qscript can involve a single GATK tool*. In that case Queue will create separate GATK commands that will each run that tool on a portion of the input data (= the scatter step). The results of each run will be stored in temporary files. Then once all the runs are done, Queue will collate all the results into the final output files, as if the tool had been run as a single command (= the gather step).

Note that Queue has additional capabilities, such as managing the use of multiple GATK tools in a dependency-aware manner to run complex pipelines, but that is outside the scope of this article. To learn more about pipelining the GATK with Queue, please see the Queue documentation.

Compare and combine

So you see, scatter-gather is a very different process from multi-threading because the parallelization happens **outside** of the program itself. The big advantage is that this opens up the upper level of computing: the cluster level. Remember, the GATK program is limited to dispatching threads to the processor of the machine on which it is run â€“ it cannot by itself send threads to a different machine. But Queue can dispatch scattered GATK jobs to different machines in a computing cluster by interfacing with your cluster's job management software.

That being said, multithreading has the great advantage that cores and machines all have access to shared machine memory with very high bandwidth capacity. In contrast, the multiple machines on a network used for scatter-gather are fundamentally limited by network costs.

The good news is that you can combine scatter-gather and multithreading: use Queue to scatter GATK jobs to different nodes on your cluster, then use the GATK's internal multithreading capabilities to parallelize the jobs running on each node.

Going back to the rice-cooking example, it's as if instead of cooking the rice yourself, you hired a catering company to do it for you. The company assigns the work to several people, who each have their own cooking station with multiple rice cookers. Now you can feed a lot more people in the same amount of time! And you don't even have to clean the dishes.

Adding Genomic Annotations Using SnpEff and VariantAnnotator #50

Last updated on 2013-06-10 18:24:01

IMPORTANT NOTE: This document is out of date and will be replaced soon. In the meantime, you can find accurate information on how to run SnpEff in a compatible way with GATK in the SnpEff documentation, and instructions on what steps are necessary in the presentation on Functional Annotation linked in the comments below.

Our testing has shown that not all combinations of snpEff/database versions produce high-quality results. Be sure to read this document completely to familiarize yourself with our recommended best practices BEFORE running snpEff.

Introduction

Until recently we were using an in-house annotation tool for genomic annotation, but the burden of keeping the database current and our lack of ability to annotate indels has led us to employ the use of a third-party tool instead. After reviewing many external tools (including annoVar, VAT, and Oncotator), we decided that SnpEff best meets our needs as it accepts VCF files as input, can annotate a full exome callset (including indels) in seconds, and provides continually-updated transcript databases. We have implemented support in the GATK for parsing the output from the SnpEff tool and annotating VCFs with the information provided in it.

SnpEff Setup and Usage

Download the SnpEff core program. If you want to be able to run VariantAnnotator on the SnpEff output, you'll need to download a version of SnpEff that VariantAnnotator supports from this page (currently supported versions are listed below). If you just want the most recent version of SnpEff and don't plan to run VariantAnnotator on its output, you can get it from here.

After unzipping the core program, open the file snpEff.config in a text editor, and change the "database_repository" line to the following:

```
database_repository = http://sourceforge.net/projects/snpeff/files/databases/
```

Then, download one or more databases using SnpEff's built-in download command:

```
java -jar snpEff.jar download GRCh37.64
```

You can find a list of available databases here. The human genome databases have **GRCh** or **hg** in their names. You can also download the databases directly from the SnpEff website, if you prefer.

The download command by default puts the databases into a subdirectory called **data** within the directory containing the SnpEff jar file. If you want the databases in a different directory, you'll need to edit the `data_dir` entry in the file `snpEff.config` to point to the correct directory.

Run SnpEff on the file containing your variants, and redirect its output to a file. SnpEff supports many input file formats including VCF 4.1, BED, and SAM pileup. Full details and command-line options can be found on the SnpEff home page.

Supported SnpEff Versions

If you want to take advantage of SnpEff integration in the GATK, you'll need to run SnpEff version **2.0.5*. Note: newer versions are currently unsupported by the GATK, as we haven't yet had the reources to test it.

Current Recommended Best Practices When Running SnpEff

These best practices are based on our analysis of various snpEff/database versions as described in detail in the **Analysis of SnpEff Annotations Across Versions** section below.

- We recommend using only the **GRCh37.64** database with SnpEff 2.0.5. The more recent GRCh37.65 database produces many false-positive Missense annotations due to a regression in the ENSEMBL Release 65 GTF file used to build the database. This regression has been acknowledged by ENSEMBL and is supposedly fixed as of 1-30-2012; however as we have not yet tested the fixed version of the database we continue to recommend using only GRCh37.64 for now.

- We recommend always running with `-onlyCoding true` with human databases (eg., the GRCh37.* databases). Setting `-onlyCoding false` causes snpEff to report all transcripts as if they were coding (even if they're not), which can lead to nonsensical results. The `-onlyCoding false` option should only be used with databases that lack protein coding information.

- Do not trust annotations from versions of snpEff prior to 2.0.4. Older versions of snpEff (such as 2.0.2)

produced many incorrect annotations due to the presence of a certain number of nonsensical transcripts in the underlying ENSEMBL databases. Newer versions of snpEff filter out such transcripts.

Analyses of SnpEff Annotations Across Versions

See our analysis of the SNP annotations produced by snpEff across various snpEff/database versions here.

- Both snpEff 2.0.2 + GRCh37.63 and snpEff 2.0.5 + GRCh37.65 produce an abnormally high Missense:Silent ratio, with elevated levels of Missense mutations across the entire spectrum of allele counts. They also have a relatively low (~70%) level of concordance with the 1000G Gencode annotations when it comes to Silent mutations. This suggests that these combinations of snpEff/database versions incorrectly annotate many Silent mutations as Missense.

- snpEff 2.0.4 RC3 + GRCh37.64 and snpEff 2.0.5 + GRCh37.64 produce a Missense:Silent ratio in line with expectations, and have a very high (~97%-99%) level of concordance with the 1000G Gencode annotations across all categories.

See our comparison of SNP annotations produced using the GRCh37.64 and GRCh37.65 databases with snpEff 2.0.5 here

- The GRCh37.64 database gives good results on the condition that you run snpEff with the `-onlyCoding true` option. The `-onlyCoding false` option causes snpEff to mark all transcripts as coding, and so produces many false-positive Missense annotations.

- The GRCh37.65 database gives results that are as poor as those you get with the `-onlyCoding false` option on the GRCh37.64 database. This is due to a regression in the ENSEMBL release 65 GTF file used to build snpEff's GRCh37.65 database. The regression has been acknowledged by ENSEMBL and is due to be fixed shortly.

See our analysis of the INDEL annotations produced by snpEff across snpEff/database versions here

- snpEff's indel annotations are highly concordant with those of a high-quality set of genomic annotations from the 1000 Genomes project. This is true across all snpEff/database versions tested.

Example SnpEff Usage with a VCF Input File

Below is an example of how to run SnpEff version 2.0.5 with a VCF input file and have it write its output in VCF format as well. Notice that you need to explicitly specify the database you want to use (in this case, GRCh37.64). This database must be present in a directory of the same name within the `data_dir` as defined in `snpEff.config`.

```
java -Xmx4G -jar snpEff.jar eff -v -onlyCoding true -i vcf -o vcf GRCh37.64 1000G.exomes.vcf
> snpEff_output.vcf
```

In this mode, SnpEff aggregates all effects associated with each variant record together into a single INFO field annotation with the key EFF. The general format is:

```
EFF=Effect1(Information about Effect1),Effect2(Information about Effect2),etc.
```

And here is the precise layout with all the subfields:

```
EFF=Effect1(Effect_Impact|Effect_Functional_Class|Codon_Change|Amino_Acid_Change|Gene_Name|Gene_BioType|Coding|Transcript_ID|Exon_ID),Effect2(etc...
```

It's also possible to get SnpEff to output in a (non-VCF) text format with one Effect per line. See the SnpEff home page for full details.

Adding SnpEff Annotations using VariantAnnotator

Once you have a SnpEff output VCF file, you can use the VariantAnnotator walker to add SnpEff annotations based on that output to the input file you ran SnpEff on.

There are two different options for doing this:

Option 1: Annotate with only the highest-impact effect for each variant

NOTE: This option works only with supported SnpEff versions as explained above. VariantAnnotator run as described below will refuse to parse SnpEff output files produced by other versions of the tool, or which lack a SnpEff version number in their header.

The default behavior when you run VariantAnnotator on a SnpEff output file is to parse the complete set of effects resulting from the current variant, select the most biologically-significant effect, and add annotations for just that effect to the INFO field of the VCF record for the current variant. This is the mode we plan to use in our Production Data-Processing Pipeline.

When selecting the most biologically-significant effect associated with the current variant, VariantAnnotator does the following:

- Prioritizes the effects according to the categories (in order of decreasing precedence) "High-Impact", "Moderate-Impact", "Low-Impact", and "Modifier", and always selects one of the effects from the highest-priority category. For example, if there are three moderate-impact effects and two high-impact effects resulting from the current variant, the annotator will choose one of the high-impact effects and add annotations based on it. See below for a full list of the effects arranged by category.

- Within each category, ties are broken using the functional class of each effect (in order of precedence: NONSENSE, MISSENSE, SILENT, or NONE). For example, if there is both a NON_SYNONYMOUS_CODING (MODERATE-impact, MISSENSE) and a CODON_CHANGE (MODERATE-impact, NONE) effect associated with the current variant, the annotator will select the NON_SYNONYMOUS_CODING effect. This is to allow for more accurate counts of the total number of sites with NONSENSE/MISSENSE/SILENT mutations. See below for a description of the functional classes SnpEff associates with the various effects.

- Effects that are within a non-coding region are always considered lower-impact than effects that are within a coding region.

Example Usage:

```
java -jar dist/GenomeAnalysisTK.jar \
```

```
    -T VariantAnnotator \
    -R /humgen/1kg/reference/human_g1k_v37.fasta \
    -A SnpEff \
    --variant 1000G.exomes.vcf \        (file to annotate)
    --snpEffFile snpEff_output.vcf \    (SnpEff VCF output file generated by running SnpEff
on the file to annotate)
    -L 1000G.exomes.vcf \
    -o out.vcf
```

VariantAnnotator adds some or all of the following INFO field annotations to each variant record:

- SNPEFF_EFFECT - The highest-impact effect resulting from the current variant (or one of the highest-impact effects, if there is a tie)
- SNPEFF_IMPACT - Impact of the highest-impact effect resulting from the current variant (HIGH, MODERATE, LOW, or MODIFIER)
- SNPEFF_FUNCTIONAL_CLASS - Functional class of the highest-impact effect resulting from the current variant (NONE, SILENT, MISSENSE, or NONSENSE)
- SNPEFF_CODON_CHANGE - Old/New codon for the highest-impact effect resulting from the current variant
- SNPEFF_AMINO_ACID_CHANGE - Old/New amino acid for the highest-impact effect resulting from the current variant
- SNPEFF_GENE_NAME - Gene name for the highest-impact effect resulting from the current variant
- SNPEFF_GENE_BIOTYPE - Gene biotype for the highest-impact effect resulting from the current variant
- SNPEFF_TRANSCRIPT_ID - Transcript ID for the highest-impact effect resulting from the current variant
- SNPEFF_EXON_ID - Exon ID for the highest-impact effect resulting from the current variant

Example VCF records annotated using SnpEff and VariantAnnotator:

```
1   874779  .   C   T   279.94  .
AC=1;AF=0.0032;AN=310;BaseQRankSum=-1.800;DP=3371;Dels=0.00;FS=0.000;HRun=0;HaplotypeScore=1
.4493;InbreedingCoeff=-0.0045;
MQ=54.49;MQ0=10;MQRankSum=0.982;QD=13.33;ReadPosRankSum=-0.060;SB=-120.09;SNPEFF_AMINO_ACID_
CHANGE=G215;SNPEFF_CODON_CHANGE=ggC/ggT;
SNPEFF_EFFECT=SYNONYMOUS_CODING;SNPEFF_EXON_ID=exon_1_874655_874840;SNPEFF_FUNCTIONAL_CLASS=
SILENT;SNPEFF_GENE_BIOTYPE=protein_coding;SNPEFF_GENE_NAME=SAMD11;
SNPEFF_IMPACT=LOW;SNPEFF_TRANSCRIPT_ID=ENST00000342066

1   874816  .   C   CT  2527.52 .
AC=15;AF=0.0484;AN=310;BaseQRankSum=-11.876;DP=4718;FS=48.575;HRun=1;HaplotypeScore=91.9147;
InbreedingCoeff=-0.0520;
MQ=53.37;MQ0=6;MQRankSum=-1.388;QD=5.92;ReadPosRankSum=-1.932;SB=-741.06;SNPEFF_EFFECT=FRAME
_SHIFT;SNPEFF_EXON_ID=exon_1_874655_874840;
SNPEFF_FUNCTIONAL_CLASS=NONE;SNPEFF_GENE_BIOTYPE=protein_coding;SNPEFF_GENE_NAME=SAMD11;SNPE
FF_IMPACT=HIGH;SNPEFF_TRANSCRIPT_ID=ENST00000342066
```

Option 2: Annotate with all effects for each variant

VariantAnnotator also has the ability to take the EFF field from the SnpEff VCF output file containing all the effects aggregated together and copy it verbatim into the VCF to annotate.

Here's an example of how to do this:

```
java -jar dist/GenomeAnalysisTK.jar \
     -T VariantAnnotator \
     -R /humgen/1kg/reference/human_g1k_v37.fasta \
     -E resource.EFF \
     --variant 1000G.exomes.vcf \      (file to annotate)
     --resource snpEff_output.vcf \    (SnpEff VCF output file generated by running SnpEff
on the file to annotate)
     -L 1000G.exomes.vcf \
     -o out.vcf
```

Of course, in this case you can also use the VCF output by SnpEff directly, but if you are using VariantAnnotator for other purposes anyway the above might be useful.

List of Genomic Effects

Below are the possible genomic effects recognized by SnpEff, grouped by biological impact. Full descriptions of each effect are available on this page.

High-Impact Effects

- SPLICE_SITE_ACCEPTOR
- SPLICE_SITE_DONOR
- START_LOST
- EXON_DELETED
- FRAME_SHIFT
- STOP_GAINED
- STOP_LOST

Moderate-Impact Effects

- NON_SYNONYMOUS_CODING
- CODON_CHANGE *(note: this effect is used by SnpEff only for MNPs, not SNPs)*
- CODON_INSERTION
- CODON_CHANGE_PLUS_CODON_INSERTION
- CODON_DELETION
- CODON_CHANGE_PLUS_CODON_DELETION

- UTR_5_DELETED
- UTR_3_DELETED

Low-Impact Effects

- SYNONYMOUS_START
- NON_SYNONYMOUS_START
- START_GAINED
- SYNONYMOUS_CODING
- SYNONYMOUS_STOP
- NON_SYNONYMOUS_STOP

Modifiers

- NONE
- CHROMOSOME
- CUSTOM
- CDS
- GENE
- TRANSCRIPT
- EXON
- INTRON_CONSERVED
- UTR_5_PRIME
- UTR_3_PRIME
- DOWNSTREAM
- INTRAGENIC
- INTERGENIC
- INTERGENIC_CONSERVED
- UPSTREAM
- REGULATION
- INTRON

Functional Classes

SnpEff assigns a functional class to certain effects, in addition to an impact:

- `NONSENSE`: assigned to point mutations that result in the creation of a new stop codon
- `MISSENSE`: assigned to point mutations that result in an amino acid change, but not a new stop codon
- `SILENT`: assigned to point mutations that result in a codon change, but not an amino acid change or new stop codon

- NONE: assigned to all effects that don't fall into any of the above categories (including all events larger than a point mutation)

The GATK prioritizes effects with functional classes over effects of equal impact that lack a functional class when selecting the most significant effect in VariantAnnotator. This is to enable accurate counts of NONSENSE/MISSENSE/SILENT sites.

BWA/C Bindings #60

Last updated on 2012-12-06 15:43:12

Sting BWA/C Bindings

WARNING: This tool is experimental and unsupported and just starting to be developed and used and should be considered a beta version. Feel free to report bugs but we are not supporting the tool

The GSA group has made bindings available for Heng Li's Burrows-Wheeler Aligner (BWA). Our aligner bindings present additional functionality to the user not traditionally available with BWA. BWA standalone is optimized to do fast, low-memory alignments from Fastq to BAM. While our bindings aim to provide support for reasonably fast, reasonably low memory alignment, we add the capacity to do exploratory data analyses. The bindings can provide all alignments for a given read, allowing a user to walk over the alignments and see information not typically provided in the BAM format. Users of the bindings can 'go deep', selectively relaxing alignment parameters one read at a time, looking for the best alignments at a site.

The BWA/C bindings should be thought of as alpha release quality. However, we aim to be particularly responsive to issues in the bindings as they arise. Because of the bindings' alpha state, some functionality is limited; see the Limitations section below for more details on what features are currently supported.

Contents

- 1 A note about using the bindings
  - 1.1 bash
  - 1.2 csh
- 2 Preparing to use the aligner
  - 2.1 Within the Broad Institute
  - 2.2 Outside of the Broad Institute
- 3 Using the existing GATK alignment walkers
- 4 Writing new GATK walkers utilizing alignment bindings
- 5 Running the aligner outside of the GATK
- 6 Limitations
- 7 Example: analysis of alignments with the BWA bindings

- 8 Validation methods

- 9 Unsupported: using the BWA/C bindings from within Matlab

A note about using the bindings

Whenever native code is called from Java, the user must assist Java in finding the proper shared library. Java looks for shared libraries in two places, on the system-wide library search path and through Java properties invoked on the command line. To add libbwa.so to the global library search path, add the following to your .my.bashrc, .my.cshrc, or other startup file:

bash

```
export LD_LIBRARY_PATH=/humgen/gsa-scr1/GATK_Data/bwa/stable:$LD_LIBRARY_PATH
```

csh

```
setenv LD_LIBRARY_PATH /humgen/gsa-scr1/GATK_Data/bwa/stable:$LD_LIBRARY_PATH
```

To specify the location of libbwa.so directly on the command-line, use the java.library.path system property as follows:

```
java -Djava.library.path=/humgen/gsa-scr1/GATK_Data/bwa/stable \
     -jar dist/GenomeAnalysisTK.jar \
     -T AlignmentValidation \
     -I /humgen/gsa-hphome1/hanna/reference/1kg/NA12878_Pilot1_20.bwa.bam \
     -R /humgen/gsa-scr1/GATK_Data/bwa/human_b36_both.fasta
```

Preparing to use the aligner

Within the Broad Institute

We provide internally accessible versions of both the BWA shared library and precomputed BWA indices for two commonly used human references at the Broad (Homo_sapiens_assembly18.fasta and human_b36_both.fasta). These files live in the following directory:

```
/humgen/gsa-scr1/GATK_Data/bwa/stable
```

Outside of the Broad Institute

Two steps are required in preparing to use the aligner: building the shared library and using BWA/C to generate an index of the reference sequence.

The Java bindings to the aligner are available through the Sting repository. A precompiled version of the bindings are available for Linux; these bindings are available in c/bwa/libbwa.so.1. To build the aligner from source:

- Fetch the latest svn of BWA from SourceForge. Configure and build BWA.

```
sh autogen.sh
./configure
make
```

- Download the latest version of Sting from our Github repository.
- Customize the variables at the top one of the build scripts (c/bwa/build_linux.sh,c/bwa/build_mac.sh) based on your environment. Run the build script.

To build a reference sequence, use the BWA C executable directly:

```
bwa index -a bwtsw <your reference sequence>.fasta
```

Using the existing GATK alignment walkers

Two walkers are provided for end users of the GATK. The first of the stock walkers is Align, which can align an unmapped BAM file or realign a mapped BAM file.

```
java \
     -Djava.library.path=/humgen/gsa-scr1/GATK_Data/bwa/stable \
     -jar dist/GenomeAnalysisTK.jar \
     -T Align \
     -I NA12878_Pilot1_20.unmapped.bam \
     -R /humgen/gsa-scr1/GATK_Data/bwa/human_b36_both.fasta \
     -U \
     -ob human.unsorted.bam
```

Most of the available parameters here are standard GATK. -T specifies that the alignment analysis should be used; -I specifies the unmapped BAM file to align, and the -R specifies the reference to which to align. By default, this walker assumes that the bwa index support files will live alongside the reference. If these files are stored elsewhere, the optional -BWT argument can be used to specify their location. By defaults, alignments will be emitted to the console in SAM format. Alignments can be spooled to disk in SAM format using the -o option or spooled to disk in BAM format using the -ob option.

The other stock walker is AlignmentValidation, which computes all possible alignments based on the BWA default configuration settings and makes sure at least one of the top alignments matches the alignment stored in the read.

```
java \
     -Djava.library.path=/humgen/gsa-scr1/GATK_Data/bwa/stable \
     -jar dist/GenomeAnalysisTK.jar \
     -T AlignmentValidation \
```

```
 -I /humgen/gsa-hphome1/hanna/reference/1kg/NA12878_Pilot1_20.bwa.bam \
 -R /humgen/gsa-scr1/GATK_Data/bwa/human_b36_both.fasta
```

Options for the AlignmentValidation walker are identical to the Alignment walker, except the AlignmentValidation walker's only output is a exception if validation fails.

Another sample walker of limited scope, CountBestAlignmentsWalker, is available for review; it is discussed in the example section below.

Writing new GATK walkers utilizing alignment bindings

BWA/C can be created on-the-fly using the org.broadinstitute.sting.alignment.bwa.c.BWACAligner constructor. The bindings have two sets of interfaces: an interface which returns all possible alignments and an interface which randomly selects an alignment from a list of the top scoring alignments as selected by BWA.

To iterate through all functions, use the following method:

```
    /**
     * Get a iterator of alignments, batched by mapping quality.
     * @param bases List of bases.
     * @return Iterator to alignments.
     */
    public Iterable<Alignment[]> getAllAlignments(final byte[] bases);
```

The call will return an Iterable which batches alignments by score. Each call to next() on the provided iterator will return all Alignments of a given score, ordered in best to worst. For example, given a read sequence with at least one match on the genome, the first call to next() will supply all exact matches, and subsequent calls to next() will give alignments judged to be inferior by BWA (alignments containing mismatches, gap opens, or gap extensions).

Alignments can be transformed to reads using the following static method in org.broadinstitute.sting.alignment.Alignment:

```
    /**
     * Creates a read directly from an alignment.
     * @param alignment The alignment to convert to a read.
     * @param unmappedRead Source of the unmapped read.  Should have bases, quality scores,
and flags.
     * @param newSAMHeader The new SAM header to use in creating this read.  Can be null,
but if so, the sequence
     *                     dictionary in the
     * @return A mapped alignment.
     */
    public static SAMRecord convertToRead(Alignment alignment, SAMRecord unmappedRead,
```

```
SAMFileHeader newSAMHeader);
```

A convenience method is available which allows the user to get SAMRecords directly from the aligner.

```
    /**
     * Get a iterator of aligned reads, batched by mapping quality.
     * @param read Read to align.
     * @param newHeader Optional new header to use when aligning the read.  If present, it
must be null.
     * @return Iterator to alignments.
     */
    public Iterable<SAMRecord[]> alignAll(final SAMRecord read, final SAMFileHeader
newHeader);
```

To return a single read randomly selected by the bindings, use one of the following methods:

```
    /**
     * Allow the aligner to choose one alignment randomly from the pile of best alignments.
     * @param bases Bases to align.
     * @return An align
     */
    public Alignment getBestAlignment(final byte[] bases);

    /**
     * Align the read to the reference.
     * @param read Read to align.
     * @param header Optional header to drop in place.
     * @return A list of the alignments.
     */
    public SAMRecord align(final SAMRecord read, final SAMFileHeader header);
```

The org.broadinstitute.sting.alignment.bwa.BWAConfiguration argument allows the user to specify parameters normally specified to 'bwt aln'. Available parameters are:

- Maximum edit distance (-n)
- Maximum gap opens (-o)
- Maximum gap extensions (-e)
- Disallow an indel within INT bp towards the ends (-i)
- Mismatch penalty (-M)
- Gap open penalty (-O)
- Gap extension penalty (-E)

Settings must be supplied to the constructor; leaving any BWAConfiguration field unset means that BWA should use its default value for that argument. Configuration settings can be updated at any time using the BWACAligner updateConfiguration method.

```
public void updateConfiguration(BWAConfiguration configuration);
```

Running the aligner outside of the GATK

The BWA/C bindings were written with running outside of the GATK in mind, but this workflow has never been tested. If you would like to run the bindings outside of the GATK, you will need:

- The BWA shared object, libbwa.so.1
- The packaged version of Aligner.jar

To build the packaged version of the aligner, run the following command

```
cp $STING_HOME/lib/bcel-*.jar ~/.ant/lib
ant package -Dexecutable=Aligner
```

This command will extract all classes required to run the aligner and place them in $STING_HOME/dist/packages/Aligner/Aligner.jar. You can then specify this one jar in your project's dependencies.

Limitations

The BWA/C bindings are currently in an alpha state, but they are extensively supported. Because of the bindings' alpha state, some functionality is limited. The limitations of these bindings include:

- Only single-end alignment is supported. However, a paired end module could be implemented as a simple extension that finds the jointly optimal placement of both singly-aligned ends.
- Color space alignments are not currently supported.
- Only a limited number of parameters BWA's extensive parameter list are supported. The current list of supported parameters is specified in the 'Writing new GATK walkers utilizing alignment bindings' section below.
- The system is not as heavily memory-optimized as the BWA/C implementation standalone. The JVM, by default, uses slightly over 4G of resident memory when running BWA on human. We have not done extensive testing on the behavior of the BWA/C bindings under memory pressure.
- There is a slight negative impact on performance when using the BWA/C bindings. BWA/C standalone on 6.9M reads of human data takes roughly 45min to run 'bwa aln', 5min to run 'bwa samse', and another 1.5min to convert the resulting SAM file to a BAM. Aligning the same dataset using the Java bindings takes approximately 55 minutes.
- The GATK requires that its input BAMs be sorted and indexed. Before using the Align or

AlignmentValidation walker, you must sort and index your unmapped BAM file. Note that this is a limitation of the GATK, not the aligner itself. Using the alignment support files outside of the GATK will eliminate this requirement.

Example: analysis of alignments with the BWA bindings

In order to validate that the Java bindings were computing the same number of reads as BWA/C standalone, we modified the BWA source to gather the number of equally scoring alignments and the frequency of the number of equally scoring alignments. We then implemented the same using a walker written in the GATK. We computed this distribution over a set of 36bp human reads and found the distributions to be identical.

The relevant parts of the walker follow.

```
public class CountBestAlignmentsWalker extends ReadWalker<Integer,Integer> {
    /**
     * The supporting BWT index generated using BWT.
     */
    @Argument(fullName="BWTPrefix",shortName="BWT",doc="Index files generated by bwa index
-d bwtsw",required=false)
    String prefix = null;

    /**
     * The actual aligner.
     */
    private Aligner aligner = null;

    private SortedMap<Integer,Integer> alignmentFrequencies = new TreeMap<Integer,Integer>
();

    /**
     * Create an aligner object.  The aligner object will load and hold the BWT until
close() is called.
     */
    @Override
    public void initialize() {
        BWTFiles bwtFiles = new BWTFiles(prefix);
        BWAConfiguration configuration = new BWAConfiguration();
        aligner = new BWACAligner(bwtFiles,configuration);
    }

    /**
     * Aligns a read to the given reference.
     * @param ref Reference over the read.  Read will most likely be unmapped, so ref will
be null.
     * @param read Read to align.
     * @return Number of alignments found for this read.
```

```
     */
    @Override
    public Integer map(char[] ref, SAMRecord read) {
        Iterator<Alignment[]> alignmentIterator =
aligner.getAllAlignments(read.getReadBases()).iterator();
        if(alignmentIterator.hasNext()) {
            int numAlignments = alignmentIterator.next().length;
            if(alignmentFrequencies.containsKey(numAlignments))

alignmentFrequencies.put(numAlignments,alignmentFrequencies.get(numAlignments)+1);
            else
                alignmentFrequencies.put(numAlignments,1);
        }
        return 1;
    }

    /**
     * Initial value for reduce.  In this case, validated reads will be counted.
     * @return 0, indicating no reads yet validated.
     */
    @Override
    public Integer reduceInit() { return 0; }

    /**
     * Calculates the number of reads processed.
     * @param value Number of reads processed by this map.
     * @param sum Number of reads processed before this map.
     * @return Number of reads processed up to and including this map.
     */
    @Override
    public Integer reduce(Integer value, Integer sum) {
        return value + sum;
    }

    /**
     * Cleanup.
     * @param result Number of reads processed.
     */
    @Override
    public void onTraversalDone(Integer result) {
        aligner.close();
        for(Map.Entry<Integer,Integer> alignmentFrequency: alignmentFrequencies.entrySet())
            out.printf("%d\t%d%n", alignmentFrequency.getKey(),
alignmentFrequency.getValue());
        super.onTraversalDone(result);
    }
```

```
}
```

This walker can be run within the svn version of the GATK using -T CountBestAlignments.

The resulting placement count frequency is shown in the graph below. The number of placements clearly follows an exponential.

Validation methods

Two major techniques were used to validate the Java bindings against the current BWA implementation.

- Fastq files from E coli and from NA12878 chr20 were aligned using BWA standalone with BWA's default settings. The aligned SAM files were sorted, indexed, and fed into the alignment validation walker. The alignment validation walker verified that one of the top scoring matches from the BWA bindings matched the alignment produced by BWA standalone.
- Fastq files from E coli and from NA12878 chr20 were aligned using the GATK Align walker, then fed back into the GATK's alignment validation walker.
- The distribution of the alignment frequency was compared between BWA standalone and the Java bindings and was found to be identical.

As an ongoing validation strategy, we will use the GATK integration test suite to align a small unmapped BAM file with human data. The contents of the unmapped BAM file will be aligned and written to disk. The md5 of the resulting file will be calculated and compared to a known good md5.

Unsupported: using the BWA/C bindings from within Matlab

Some users are attempting to use the BWA/C bindings from within Matlab. To run the GATK within Matlab, you'll need to add libbwa.so to your library path through the librarypath.txt file. The librarypath.txt file normally lives in $matlabroot/toolbox/local. Within the Broad Institute, the $matlabroot/toolbox/local/librarypath.txt file is shared; therefore, you'll have to create a librarypath.txt file in your working directory from which you execute matlab.

```
##
## FILE: librarypath.txt
##
## Entries:
##    o path_to_jnifile
##    o [alpha,glnx86,sol2,unix,win32,mac]=path_to_jnifile
##    o $matlabroot/path_to_jnifile
##    o $jre_home/path_to_jnifile
##
$matlabroot/bin/$arch
```

```
/humgen/gsa-scr1/GATK_Data/bwa/stable
```

Once you've edited the library path, you can verify that Matlab has picked up your modified file by running the following command:

```
>> java.lang.System.getProperty('java.library.path')

ans =
/broad/tools/apps/matlab2009b/bin/glnxa64:/humgen/gsa-scr1/GATK_Data/bwa/stable
```

Once the location of libbwa.so has been added to the library path, you can use the BWACAligner just as you would any other Java class in Matlab:

```
>> javaclasspath({'/humgen/gsa-scr1/hanna/src/Sting/dist/packages/Aligner/Aligner.jar'})
>> import org.broadinstitute.sting.alignment.bwa.BWTFiles
>> import org.broadinstitute.sting.alignment.bwa.BWAConfiguration
>> import org.broadinstitute.sting.alignment.bwa.c.BWACAligner
>> x =
BWACAligner(BWTFiles('/humgen/gsa-scr1/GATK_Data/bwa/Homo_sapiens_assembly18.fasta'),BWAConf
iguration())
>> y=x.getAllAlignments(uint8('CCAATAACCAAGGCTGTTAGGTATTTTATCAGCAATGTGGGATAAGCAC'));
```

We don't have the resources to directly support using the BWA/C bindings from within Matlab, but if you report problems to us, we will try to address them.

Base Quality Score Recalibration (BQSR) #44

Last updated on 2013-09-05 16:22:33

Detailed information about command line options for BaseRecalibrator can be found here.

Introduction

The tools in this package recalibrate base quality scores of sequencing-by-synthesis reads in an aligned BAM file. After recalibration, the quality scores in the QUAL field in each read in the output BAM are more accurate in that the reported quality score is closer to its actual probability of mismatching the reference genome. Moreover, the recalibration tool attempts to correct for variation in quality with machine cycle and sequence context, and by doing so provides not only more accurate quality scores but also more widely dispersed ones. The system works on BAM files coming from many sequencing platforms: Illumina, SOLiD, 454, Complete Genomics, Pacific Biosciences, etc.

New with the release of the full version of GATK 2.0 is the ability to recalibrate not only the well-known base quality scores but also base insertion and base deletion quality scores. These are per-base quantities which

estimate the probability that the next base in the read was mis-incorporated or mis-deleted (due to slippage, for example). We've found that these new quality scores are very valuable in indel calling algorithms. In particular these new probabilities fit very naturally as the gap penalties in an HMM-based indel calling algorithms. We suspect there are many other fantastic uses for these data.

This process is accomplished by analyzing the covariation among several features of a base. For example:

- Reported quality score
- The position within the read
- The preceding and current nucleotide (sequencing chemistry effect) observed by the sequencing machine

These covariates are then subsequently applied through a piecewise tabular correction to recalibrate the quality scores of all reads in a BAM file.

For example, pre-calibration a file could contain only reported Q25 bases, which seems good. However, it may be that these bases actually mismatch the reference at a 1 in 100 rate, so are actually Q20. These higher-than-empirical quality scores provide false confidence in the base calls. Moreover, as is common with sequencing-by-synthesis machine, base mismatches with the reference occur at the end of the reads more frequently than at the beginning. Also, mismatches are strongly associated with sequencing context, in that the dinucleotide AC is often much lower quality than TG. The recalibration tool will not only correct the average Q inaccuracy (shifting from Q25 to Q20) but identify subsets of high-quality bases by separating the low-quality end of read bases AC bases from the high-quality TG bases at the start of the read. See below for examples of pre and post corrected values.

The system was designed for users to be able to easily add new covariates to the calculations. For users wishing to add their own covariate simply look at QualityScoreCovariate.java for an idea of how to implement the required interface. Each covariate is a Java class which implements the org.broadinstitute.sting.gatk.walkers.recalibration.Covariate interface. Specifically, the class needs to have a getValue method defined which looks at the read and associated sequence context and pulls out the desired information such as machine cycle.

Running the tools

BaseRecalibrator

Detailed information about command line options for BaseRecalibrator can be found here.

This GATK processing step walks over all of the reads in `my_reads.bam` and tabulates data about the following features of the bases:

- read group the read belongs to
- assigned quality score
- machine cycle producing this base
- current base + previous base (dinucleotide)

For each bin, we count the number of bases within the bin and how often such bases mismatch the reference base, excluding loci known to vary in the population, according to dbSNP. After running over all reads, BaseRecalibrator produces a file called `my_reads.recal_data.grp`, which contains the data needed to recalibrate reads. The format of this GATK report is described below.

Creating a recalibrated BAM

To create a recalibrated BAM you can use GATK's PrintReads with the engine on-the-fly recalibration capability. Here is a typical command line to do so:

```
java -jar GenomeAnalysisTK.jar \
     -T PrintReads \
     -R reference.fasta \
     -I input.bam \
     -BQSR recalibration_report.grp \
     -o output.bam
```

After computing covariates in the initial BAM File, we then walk through the BAM file again and rewrite the quality scores (in the QUAL field) using the data in the `recalibration_report.grp` file, into a new BAM file.

This step uses the recalibration table data in recalibration_report.grp produced by BaseRecalibration to recalibrate the quality scores in input.bam, and writing out a new BAM file output.bam with recalibrated QUAL field values.

Effectively the new quality score is:

- the sum of the global difference between reported quality scores and the empirical quality
- plus the quality bin specific shift
- plus the cycle x qual and dinucleotide x qual effect

Following recalibration, the read quality scores are much closer to their empirical scores than before. This means they can be used in a statistically robust manner for downstream processing, such as SNP calling. In additional, by accounting for quality changes by cycle and sequence context, we can identify truly high quality bases in the reads, often finding a subset of bases that are Q30 even when no bases were originally labeled as such.

Miscellaneous information

- The recalibration system is read-group aware. It separates the covariate data by read group in the recalibration_report.grp file (using @RG tags) and PrintReads will apply this data for each read group in the file. We routinely process BAM files with multiple read groups. Please note that the memory requirements scale linearly with the number of read groups in the file, so that files with many read groups could require a significant amount of RAM to store all of the covariate data.
- A critical determinant of the quality of the recalibation is the number of observed bases and mismatches in

each bin. The system will not work well on a small number of aligned reads. We usually expect well in excess of 100M bases from a next-generation DNA sequencer per read group. 1B bases yields significantly better results.

- Unless your database of variation is so poor and/or variation so common in your organism that most of your mismatches are real snps, you should always perform recalibration on your bam file. For humans, with dbSNP and now 1000 Genomes available, almost all of the mismatches - even in cancer - will be errors, and an accurate error model (essential for downstream analysis) can be ascertained.

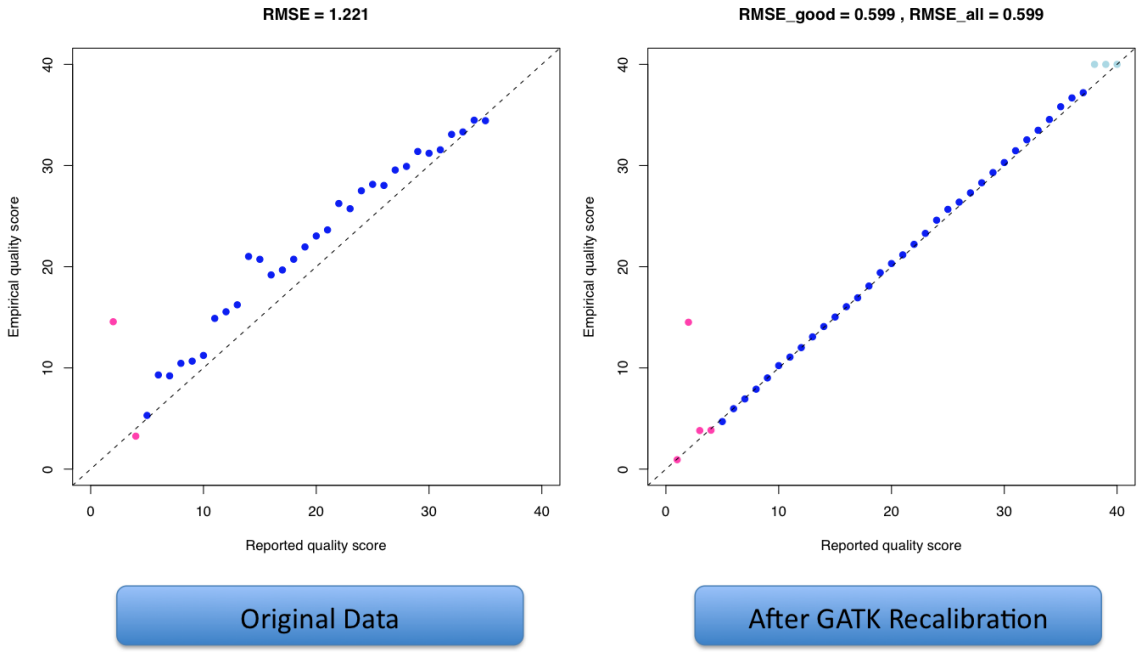
- The recalibrator applies a "yates" correction for low occupancy bins. Rather than inferring the true Q score from # mismatches / # bases we actually infer it from (# mismatches + 1) / (# bases + 2). This deals very nicely with overfitting problems, which has only a minor impact on data sets with billions of bases but is critical to avoid overconfidence in rare bins in sparse data.

Example pre and post recalibration results

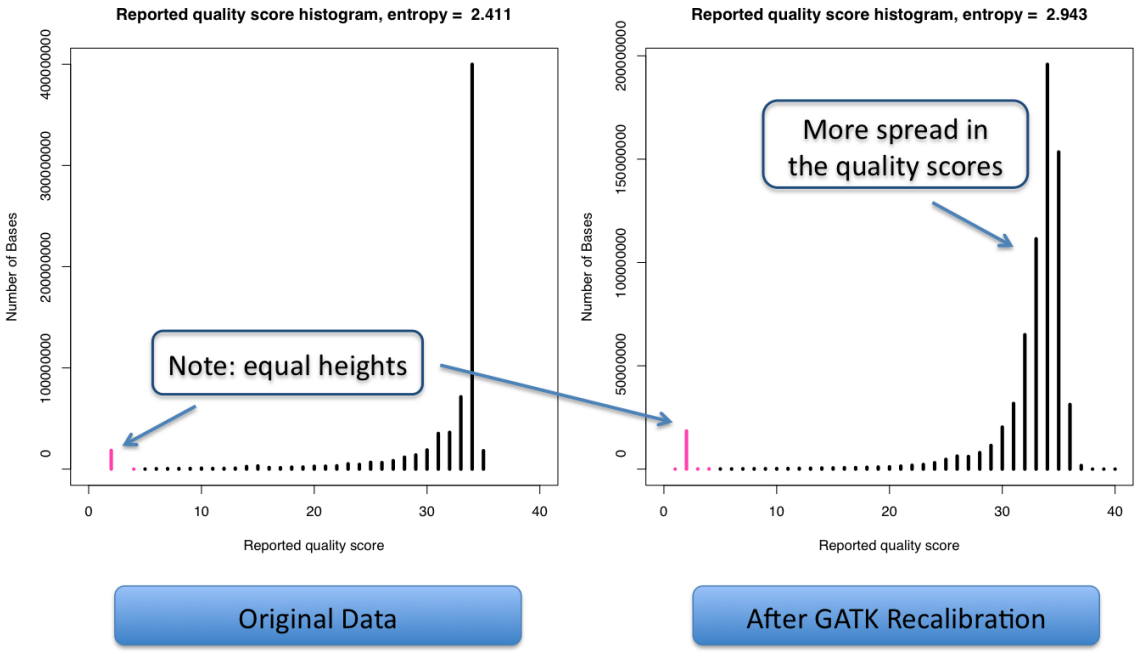
- Recalibration of a lane sequenced at the Broad by an Illumina GA-II in February 2010

- There is a significant improvement in the accuracy of the base quality scores after applying the GATK recalibration procedure

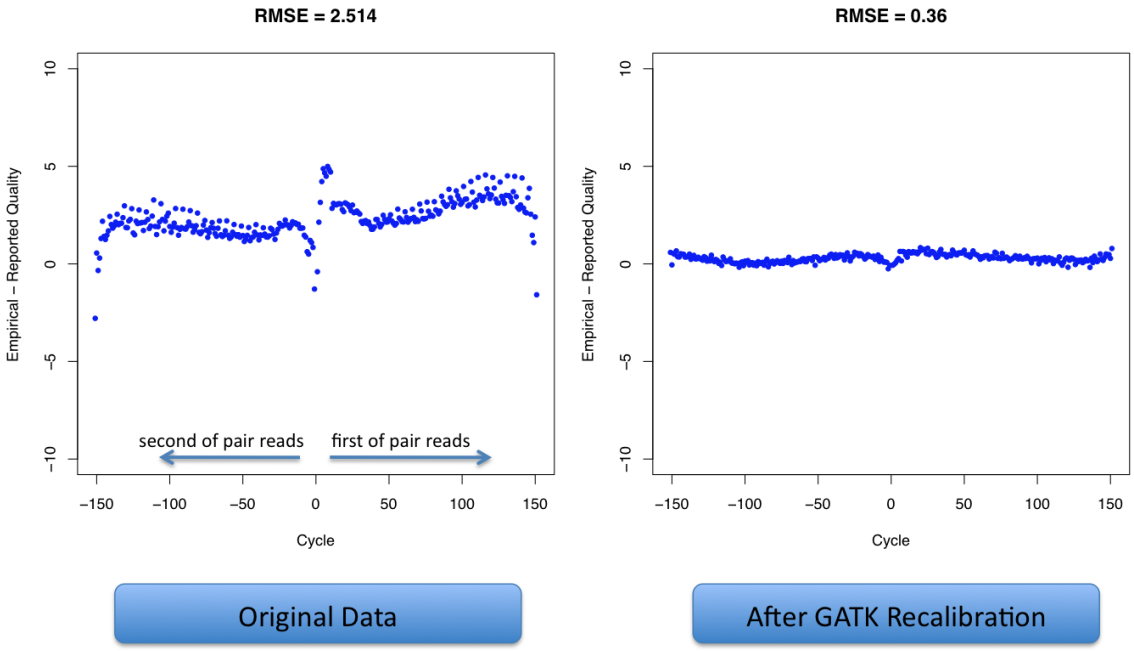
Reported Quality vs. Empirical Quality



Distribution of Quality Scores

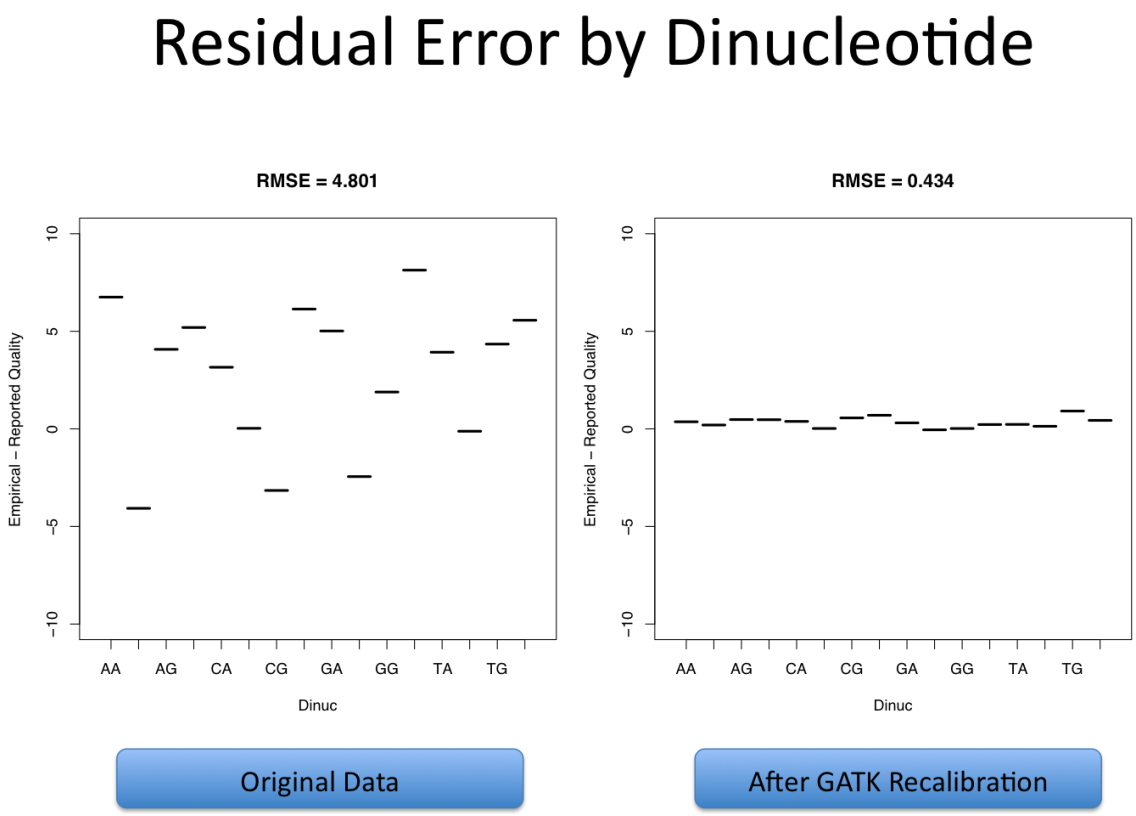


Residual Error by Machine Cycle



Original Data

After GATK Recalibration



The output of the BaseRecalibrator

- A Recalibration report containing all the recalibration information for the data

Note that the BasRecalibrator no longer produces plots; this is now done by the AnalyzeCovariates tool.

The Recalibration Report

The recalibration report is a [GATKReport](http://gatk.vanillaforums.com/discussion/1244/what-is-a-gatkreport) and not only contains the main result of the analysis, but it is also used as an input to all subsequent analyses on the data. The recalibration report contains the following 5 tables:

- Arguments Table -- a table with all the arguments and its values
- Quantization Table
- ReadGroup Table
- Quality Score Table

- Covariates Table

Arguments Table

This is the table that contains all the arguments used to run BQSRv2 for this dataset. This is important for the on-the-fly recalibration step to use the same parameters used in the recalibration step (context sizes, covariates, ...).

Example Arguments table:

```
#:GATKTable:true:1:17::;
#:GATKTable:Arguments:Recalibration argument collection values used in this run
Argument                    Value
covariate                   null
default_platform            null
deletions_context_size      6
force_platform              null
insertions_context_size     6
...
```

Quantization Table

The GATK offers native support to quantize base qualities. The GATK quantization procedure uses a statistical approach to determine the best binning system that minimizes the error introduced by amalgamating the different qualities present in the specific dataset. When running BQSRv2, a table with the base counts for each base quality is generated and a 'default' quantization table is generated. This table is a required parameter for any other tool in the GATK if you want to quantize your quality scores.

The default behavior (currently) is to use no quantization when performing on-the-fly recalibration. You can override this by using the engine argument -qq. With -qq 0 you don't quantize qualities, or -qq N you recalculate the quantization bins using N bins on the fly. Note that quantization is completely experimental now and we do not recommend using it unless you are a super advanced user.

Example Arguments table:

```
#:GATKTable:true:2:94:::;
#:GATKTable:Quantized:Quality quantization map
QualityScore  Count        QuantizedScore
0                      252               0
1                    15972               1
2                   553525               2
3                  2190142               9
4                  5369681               9
9                 83645762               9
...
```

ReadGroup Table

This table contains the empirical quality scores for each read group, for mismatches insertions and deletions. This is not different from the table used in the old table recalibration walker.

```
#:GATKTable:false:6:18:%s:%s:%.4f:%.4f:%d:%d:;
#:GATKTable:RecalTable0:
ReadGroup  EventType  EmpiricalQuality  EstimatedQReported  Observations  Errors
SRR032768  D                  40.7476             45.0000    2642683174    222475
SRR032766  D                  40.9072             45.0000    2630282426    213441
SRR032764  D                  40.5931             45.0000    2919572148    254687
SRR032769  D                  40.7448             45.0000    2850110574    240094
SRR032767  D                  40.6820             45.0000    2820040026    241020
SRR032765  D                  40.9034             45.0000    2441035052    198258
SRR032766  M                  23.2573             23.7733    2630282426  12424434
SRR032768  M                  23.0281             23.5366    2642683174  13159514
SRR032769  M                  23.2608             23.6920    2850110574  13451898
SRR032764  M                  23.2302             23.6039    2919572148  13877177
SRR032765  M                  23.0271             23.5527    2441035052  12158144
SRR032767  M                  23.1195             23.5852    2820040026  13750197
SRR032766  I                  41.7198             45.0000    2630282426    177017
SRR032768  I                  41.5682             45.0000    2642683174    184172
SRR032769  I                  41.5828             45.0000    2850110574    197959
SRR032764  I                  41.2958             45.0000    2919572148    216637
SRR032765  I                  41.5546             45.0000    2441035052    170651
SRR032767  I                  41.5192             45.0000    2820040026    198762
```

Quality Score Table

This table contains the empirical quality scores for each read group and original quality score, for mismatches insertions and deletions. This is not different from the table used in the old table recalibration walker.

```
#:GATKTable:false:6:274:%s:%s:%s:%.4f:%d:%d:;
#:GATKTable:RecalTable1:
ReadGroup  QualityScore  EventType  EmpiricalQuality  Observations  Errors
SRR032767            49  M                  33.7794          9549        3
SRR032769            49  M                  36.9975          5008        0
SRR032764            49  M                  39.2490          8411        0
SRR032766            18  M                  17.7397      16330200   274803
SRR032768            18  M                  17.7922      17707920   294405
SRR032764            45  I                  41.2958    2919572148   216637
SRR032765             6  M                   6.0600       3401801   842765
SRR032769            45  I                  41.5828    2850110574   197959
SRR032764             6  M                   6.0751       4220451  1041946
SRR032767            45  I                  41.5192    2820040026   198762
```

```
SRR032769             6  M                   6.3481       5045533  1169748
SRR032768            16  M                  15.7681      12427549   329283
SRR032766            16  M                  15.8173      11799056   309110
SRR032764            16  M                  15.9033      13017244   334343
SRR032769            16  M                  15.8042      13817386   363078
...
```

Covariates Table

This table has the empirical qualities for each covariate used in the dataset. The default covariates are cycle and context. In the current implementation, context is of a fixed size (default 6). Each context and each cycle will have an entry on this table stratified by read group and original quality score.

```
#:GATKTable:false:8:1003738:%s:%s:%s:%s:%s:%.4f:%d:%d:;
#:GATKTable:RecalTable2:
ReadGroup  QualityScore  CovariateValue  CovariateName  EventType  EmpiricalQuality
Observations  Errors
SRR032767            16  TACGGA          Context        M                  14.2139
 817      30
SRR032766            16  AACGGA          Context        M                  14.9938
1420      44
SRR032765            16  TACGGA          Context        M                  15.5145
 711      19
SRR032768            16  AACGGA          Context        M                  15.0133
1585      49
SRR032764            16  TACGGA          Context        M                  14.5393
 710      24
SRR032766            16  GACGGA          Context        M                  17.9746
1379      21
SRR032768            45  CACCTC          Context        I                  40.7907
575849      47
SRR032764            45  TACCTC          Context        I                  43.8286
507088      20
SRR032769            45  TACGGC          Context        D                  38.7536
37525       4
SRR032768            45  GACCTC          Context        I                  46.0724
445275      10
SRR032766            45  CACCTC          Context        I                  41.0696
575664      44
SRR032769            45  TACCTC          Context        I                  43.4821
490491      21
SRR032766            45  CACGGC          Context        D                  45.1471
65424       1
SRR032768            45  GACGGC          Context        D                  45.3980
34657       0
SRR032767            45  TACGGC          Context        D                  42.7663
```

```
37814       1
SRR032767            16  AACGGA          Context        M                    15.9371
1647      41
SRR032764            16  GACGGA          Context        M                    18.2642
1273      18
SRR032769            16  CACGGA          Context        M                    13.0801
1442      70
SRR032765            16  GACGGA          Context        M                    15.9934
1271      31
...
```

Troubleshooting

The memory requirements of the recalibrator will vary based on the type of JVM running the application and the number of read groups in the input bam file.

If the application reports 'java.lang.OutOfMemoryError: Java heap space', increase the max heap size provided to the JVM by adding ' -Xmx????m' to the jvm_args variable in RecalQual.py, where '????' is the maximum available memory on the processing computer.

I've tried recalibrating my data using a downloaded file, such as NA12878 on 454, and apply the table to any of the chromosome BAM files always fails due to hitting my memory limit. I've tried giving it as much as 15GB but that still isn't enough.

All of our big merged files for 454 are running with -Xmx16000m arguments to the JVM -- it's enough to process all of the files. 32GB might make the 454 runs a lot faster though.

I have a recalibration file calculated over the entire genome (such as for the 1000 genomes trio) but I split my file into pieces (such as by chromosome). Can the recalibration tables safely be applied to the per chromosome BAM files?

Yes they can. The original tables needed to be calculated over the whole genome but they can be applied to each piece of the data set independently.

I'm working on a genome that doesn't really have a good SNP database yet. I'm wondering if it still makes sense to run base quality score recalibration without known SNPs.

The base quality score recalibrator treats every reference mismatch as indicative of machine error. True polymorphisms are legitimate mismatches to the reference and shouldn't be counted against the quality of a base. We use a database of known polymorphisms to skip over most polymorphic sites. Unfortunately without this information the data becomes almost completely unusable since the quality of the bases will be inferred to be much much lower than it actually is as a result of the reference-mismatching SNP sites.

However, all is not lost if you are willing to experiment a bit. You can bootstrap a database of known SNPs. Here's how it works:

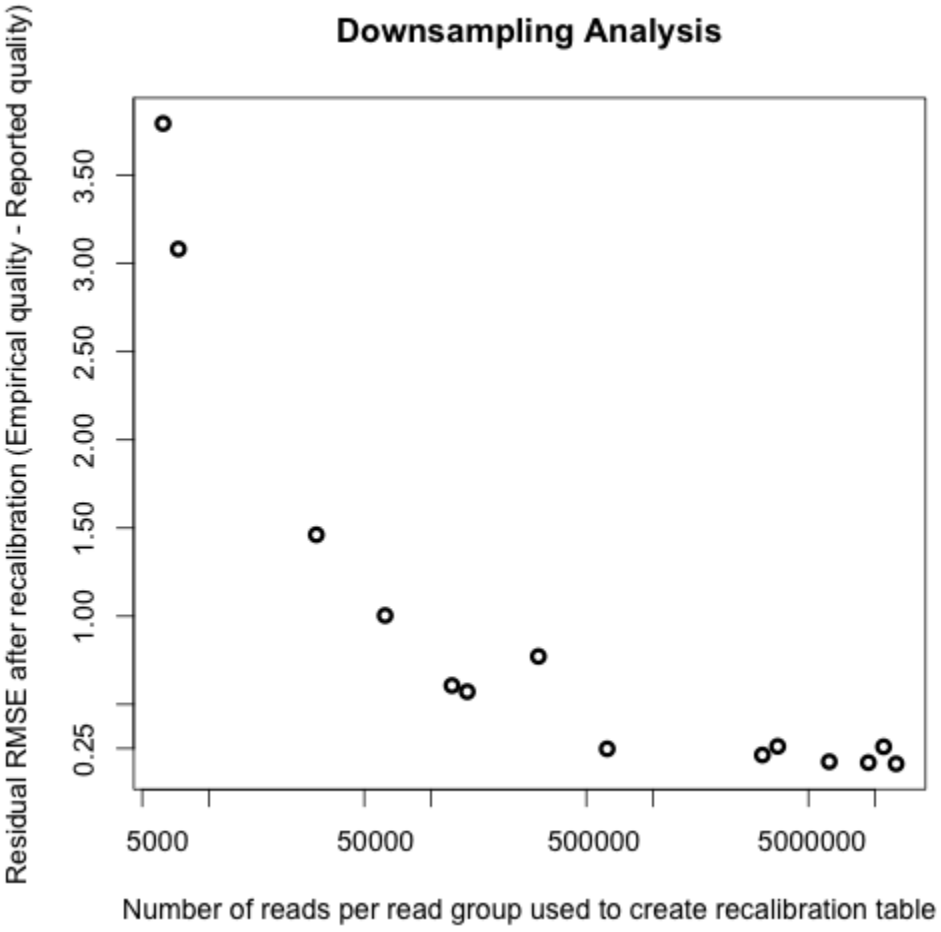
- First do an initial round of SNP calling on your original, unrecalibrated data.

- Then take the SNPs that you have the highest confidence in and use that set as the database of known SNPs by feeding it as a VCF file to the base quality score recalibrator.

- Finally, do a real round of SNP calling with the recalibrated data. These steps could be repeated several times until convergence.

Downsampling to reduce run time

For users concerned about run time please note this small analysis below showing the approximate number of reads per read group that are required to achieve a given level of recalibration performance. The analysis was performed with 51 base pair Illumina reads on pilot data from the 1000 Genomes Project. Downsampling can be achieved by specifying a genome interval using the -L option. For users concerned only with recalibration accuracy please disregard this plot and continue to use all available data when generating the recalibration table.



Best Practices for Variant Detection with GATK #3238

Last updated on 2013-09-13 21:04:54

The Best Practices have been updated for GATK version 2.7. If you are running an older version, you should seriously consider upgrading. For more details about what has changed in each version, please see the Version History section. If you cannot upgrade your version of GATK for any reason, please look up the corresponding version of the GuideBook PDF (also in the Version History section) to ensure that you are using the appropriate recommendations for your version.

Companion Utilities: ReorderSam #58

Last updated on 2013-03-25 19:13:51

ReorderSam

The GATK can be particular about the ordering of a BAM file. If you find yourself in the not uncommon situation of having created or received BAM files sorted in a bad order, you can use the tool ReorderSam to generate a new BAM file where the reads have been reordered to match a well-ordered reference file.

```
java -jar picard/ReorderSam.jar I= lexicographc.bam O= kayrotypic.bam REFERENCE=
Homo_sapiens_assembly18.kayrotypic.fasta
```

This tool requires you have a correctly sorted version of the reference sequence you used to align your reads. This tool will drop reads that don't have equivalent contigs in the new reference (potentially bad, but maybe not). If contigs have the same name in the bam and the new reference, this tool assumes that the alignment of the read in the new BAM is the same. This is not a lift over tool!

The tool, though once in the GATK, is now part of the Picard package.

Companion Utilities: ReplaceReadGroups #59

Last updated on 2012-09-28 16:29:19

This utility replaces read groups in a BAM file

It is useful for fixing problems such as not having read groups in a bam file.

```
java -jar picard/AddOrReplaceReadGroups.jar I= testdata/exampleNORG.bam O= exampleNewRG.bam
SORT_ORDER=coordinate RGID=foo RGLB=bar RGPL=illumina RGSM=DePristo
```

Note that this tool is now part of the Picard package:
http://picard.sourceforge.net/command-line-overview.shtml#AddOrReplaceReadGroups

This tool can fix BAM files without read group information:

```
# throws an error
java -jar dist/GenomeAnalysisTK.jar -R testdata/exampleFASTA.fasta -I
```

```
testdata/exampleNORG.bam -T UnifiedGenotyper

# fix the read groups
java -jar picard/AddOrReplaceReadGroups.jar I= testdata/exampleNORG.bam O= exampleNewRG.bam
SORT_ORDER=coordinate RGID=foo RGLB=bar RGPL=illumina RGSM=DePristo CREATE_INDEX=True

# runs without error
java -jar dist/GenomeAnalysisTK.jar -R testdata/exampleFASTA.fasta -I exampleNewRG.bam -T
UnifiedGenotyper
```

Creating Amplicon Sequences #57

Last updated on 2012-09-28 16:56:14

Creating Amplicon Sequences

Note that earlier versions of the GATK used a different tool.

For a complete, detailed argument reference, refer to the GATK document page here

Contents

- 1 Introduction
- 1.1 Lowercase and Ns
- 1.2 BWA Bindings

- 2 Running Validation Amplicons
- 3 Validation Amplicons Output
- 4 Warnings During Traversal

Introduction

This tool generates amplicon sequences for use with the Sequenom primer design tool. The output of this tool is fasta-formatted, where the characters [A/B] specify the allele to be probed (see Validation Amplicons Output further below). It can mask nearby variation (either by 'N' or by lower-casing characters), and can try to restrict sequenom design to regions of the amplicon likely to generate a highly specific primer. This tool will also flag sites with properties that could shift the mass-spec peak from its expected value, such as indels in the amplicon sequence, SNPs within 4 bases of the variant attempting to be probed, or multiple variants selected for validation falling into the same amplicon.

Lowercase and Ns

Ns in the amplicon sequence instructs primer design software (such as Sequenom) not to use that base in the primer: any primer will fall entirely before, or entirely after, that base. Lower-case letters instruct the design

software to try to avoid using the base (presumably by applying a penalty for doing so), but will not prevent it from doing so if a good primer (i.e. a primer with suitable melting temperature and low probability of hairpin formation) is found.

BWA Bindings

ValidationAmplicons relies on the GATK Sting BWA/C bindings to assess the specificity of potential primers. The wiki page for Sting BWA/C bindings contains **required information** about how to download the appropriate version of BWA, how to create a BWT reference, and how to set your classpath appropriately to run this tool. **If you have not followed the directions to set up the BWA/C bindings, you will not be able to create validation amplicon sequences using the GATK**. There is an argument (see below) to disable the use of BWA, and lower repeats within the amplicon only. Use of this argument is not recommended.

Running Validation Amplicons

Validation Amplicons **requires** three input files: a VCF of alleles you want to validate, a VCF of variants you want to mask, and a Table of intervals around the variants describing the size of the amplicons. For instance:

Alleles to Validate

```
##fileformat=VCFv4.0
#CHROM      POS     ID      REF     ALT     QUAL    FILTER  INFO
20          207414  .       G    A       85.09   PASS    .   // SNP to validate
20          792122  .    TCCC    T       22.24   PASS    .   // DEL to validate
20          994145  .       G    GAAG    48.21   PASS    .   // INS to validate
20          1074230 .       C    T       2.29     QD     .   // SNP to validate (but
filtered)
20          1084330 .      AC   GT       42.21   PASS    .   // MNP to validate
```

Interval Table

```
HEADERpos               name
20:207334-207494        20_207414
20:792042-792202        20_792122
20:994065-994225        20_994145
20:1074150-1074310      20_1074230
20:1084250-1084410      20_1084330
```

Alleles to Mask

```
##fileformat=VCFv4.1
#CHROM  POS      ID        REF         ALT     QUAL       FILTER         INFO
20      207414   .         G           A       77.12      PASS            .
20      207416   .         A           AGGC    49422.34   PASS            .
```

```
20      792076   .         A           G       2637.15    HaplotypeScore  .
20      792080   .         T           G       161.83     PASS            .
20      792087   .         CGGT        C       179.84     ReadPosRankSum  .
20      792106   .         C           G       32.59      PASS            .
20      792140   .         C           G       409.75     PASS            .
20      1084319  .         T           A,C     22.24      PASS            .
20      1084348  .  TACCACCCCACACA     T      482.84      PASS            .
```

Validation Amplicons Output

The output from Validation Amplicons is a fasta-formatted file, with a small adaptation to represent the site being probed. Using the test files above, the output of the command

```
java -jar $GATK/dist/GenomeAnalysisTK.jar \
     -T ValidationAmplicons \
     -R /humgen/1kg/reference/human_g1k_v37.fasta \
     -BTI ProbeIntervals \
     --ProbeIntervals:table interval_table.table \
     --ValidateAlleles:vcf sites_to_validate.vcf \
     --MaskAlleles:vcf mask_sites.vcf \
     --virtualPrimerSize 30 \
     -o probes.fasta \
     -l WARN
```

is

```
>20:207414 INSERTION=1,VARIANT_TOO_NEAR_PROBE=1, 20_207414
CCAACGTTAAGAAAGAGACATGCGACTGGGTgcggtggctcatgcctggaaccccagcactttgggaggccaaggtgggc[A/G*]gNNcac
ttgaggtcaggagtttgagaccagcctggccaacatggtgaaaccccgtctctactgaaaatacaaaagttagC
>20:792122 Valid 20_792122
TTTTTTTTTagatggagtctcgctcttatcgcccaggcNggagtgggtggtgtgatcttggctNactgcaacttctgcct[-/CCC*]ccca
ggttcaagtgattNtcctgcctcagccacctgagtagctgggattacaggcatccgccaccatgcctggctaatTT
>20:994145 Valid 20_994145
TCCATGGCCTCCCCCTGGCCCACGAAGTCCTCAGCCACCTCCTTCCTGGAGGGCTCAGCCAAAATCAGACTGAGGAAGAAG[AAG/-*]TGG
TGGGCACCCACCTTCTGGCCTTCCTCAGCCCCTTATTCCTAGGACCAGTCCCCATCTAGGGGTCCTCACTGCCTCCC
>20:1074230 SITE_IS_FILTERED=1, 20_1074230
ACCTGATTACCATCAATCAGAACTCATTTCTGTTCCTATCTTCCACCCACAATTGTAATGCCTTTTCCATTTTAACCAAG[T/C*]ACTTAT
TATAtactatggccataacttttgcagtttgaggtatgacagcaaaaTTAGCATACATTTCATTTTCCTTCTTC
>20:1084330 DELETION=1, 20_1084330
CACGTTCGGcttgtgcagagcctcaaggtcatccagaggtgatAGTTTAGGGCCCTCTCAAGTCTTTCCNGTGCGCATGG[GT/AC*]CAGC
CCTGGGCACCTGTNNNNNNNNNNNNNTGCTCATGGCCTTCTAGATTCCCAGGAAATGTCAGAGCTTTTCAAAGCCC
```

Note that SNPs have been masked with 'N's, filtered 'mask' variants do not appear, the insertion has been flanked by Ns, the unfiltered deletion has been replaced by Ns, and the filtered site in the validation VCF is not

marked as valid. In addition, bases that fall inside at least one non-unique 30-mer (meaning no multiple MQ0 alignments using BWA) are lower-cased. The identifier for each sequence is the position of the allele to be probed, a 'validation status' (defined below), and a string representing the amplicon. Validation status values are:

```
Valid                     // amplicon is valid
SITE_IS_FILTERED=1        // validation site is not marked 'PASS' or '.' in its filter field
("you are trying to validate a filtered variant")
VARIANT_TOO_NEAR_PROBE=1  // there is a variant too near to the variant to be validated,
potentially shifting the mass-spec peak
MULTIPLE_PROBES=1,        // multiple variants to be validated found inside the same
amplicon
DELETION=6,INSERTION=5,   // 6 deletions and 5 insertions found inside the amplicon region
(from the "mask" VCF), will be potentially difficult to validate
DELETION=1,               // deletion found inside the amplicon region, could shift
mass-spec peak
START_TOO_CLOSE,          // variant is too close to the start of the amplicon region to
give sequenom a good chance to find a suitable primer
END_TOO_CLOSE,            // variant is too close to the end of the amplicon region to give
sequenom a good chance to find a suitable primer
NO_VARIANTS_FOUND,        // no variants found within the amplicon region
INDEL_OVERLAPS_VALIDATION_SITE, // an insertion or deletion interferes directly with the
site to be validated (i.e. insertion directly preceding or postceding, or a deletion that
spans the site itself)
```

Warnings During Traversal

The files provided to Validation Amplicons should be such that all generated amplicons are valid. That means:

```
There are no variants within 4bp of the site to be validated
There are no indels in the amplicon region
Amplicon windows do not include other sites to be probed
Amplicon windows are not too short, and the variant therein is not within 50bp of either
edge
All amplicon windows contain a variant to be validated
Variants to be validated are unfiltered or pass filters
```

The tool will warn you each time any of these conditions are not met.

Creating Variant Validation Sets #55

Last updated on 2012-09-28 17:49:21

Contents

- 1 Introduction
- 2 GATK Documentation
- 3 Sample and Frequency Restrictions
- 3.1 -sampleMode
- 3.2 -samplePNonref
- 3.3 -frequencySelectionMode

Introduction

ValidationSiteSelectorWalker is intended for use in experiments where we sample data randomly from a set of variants, for example in order to choose sites for a follow-up validation study. Sites are selected randomly but within certain restrictions. There are two main sources of restrictions: Sample restrictions and Frequency restrictions. Sample restrictions alter the polymorphic/monomorphic status of sites by restricting the sample set to a given number of samples. Frequency restrictions bias the site sampling method to sample either uniformly, or in accordance with the allele frequency spectrum of the input VCF.

GATK Documentation

For example command lines and a full list of arguments, please see the GATK documentation for this tool at Validation Site Selector.

Sample and Frequency Restrictions

-sampleMode

The -sampleMode argument controls the mode of sample-based site consideration. The options are:

- None: All sites are included for consideration, including reference sites
- Poly_based_on_gt: Site is included if it has a variant genotype in at least one of the selected samples
- Poly_based_on_gl: Site is included if it is likely to be variant based on the genotype likelihoods of the selected samples

-samplePNonref

Note that Poly_based_on_gl uses the exact allele frequency calculation model to estimate P[site is nonref]. The site is considered for validation if P[site is nonref] > [this argument]. So if you want to validate sites that are > 95% confidently nonref (based on the likelihoods), you would set -sampleMode POLY_BASED_ON_GL -samplePNonref 0.95

-frequencySelectionMode

The -frequencySelectionMode argument controls the mode of frequency matching for site selection. The options are:

- Uniform: Choose variants uniformly, without regard to their allele frequency.
- Keep AF Spectrum: Choose variants so that the resulting allele frequency matches as closely as possible to that of the input VCF.

Data Processing Pipeline [RETIRED] #41

Last updated on 2013-09-13 17:52:31

Please note that the DataProcessingPipeline qscript is no longer available.

The DPP script was only provided has an example, but many people were using it "out of the box" without properly understanding how it works. In order to protect users from mishandling this tool, and to decrease our support burden, we have taken the difficult decision of removing the script from our public repository. If you would like to put together your own version of the DPP, please have a look at our other example scripts to understand how Qscripts work, and read the Best Practices documentation to understand what are the processing steps and what parameters you need to set/adjust.

Data Processing Pipeline

The Data Processing Pipeline is a Queue script designed to take BAM files from the NGS machines to analysis ready BAMs for the GATK.

Introduction

Reads come off the sequencers in a raw state that is not suitable for analysis using the GATK. In order to prepare the dataset, one must perform the steps described here. This pipeline performs the following steps: indel cleaning, duplicate marking and base score recalibration, following the GSA's latest definition of best practices. The product of this pipeline is a set of analysis ready BAM files (one per sample sequenced).

Requirements

This pipeline is a Queue script that uses tools from the GATK, Picard and BWA (optional) software suites which are all freely available through their respective websites. Queue is a GATK companion that is included in the GATK package.

Warning: This pipeline was designed specifically to handle the Broad Institute's main sequencing pipeline with Illumina BAM files and BWA alignment. The GSA cannot support its use for other types of datasets. It is possible however, with some effort, to modify it for your needs.

Command-line arguments

Required Parameters

Argument (short-name) Argument (long-name) Description -i <BAM file / BAM list> --input <BAM file / BAM list> input BAM file - or list of BAM files. -R <fasta> --reference <fasta> Reference fasta file. -D <vcf > --dbsnp <dbsnp vcf> dbsnp ROD to use (must be in VCF format).

Optional Parameters

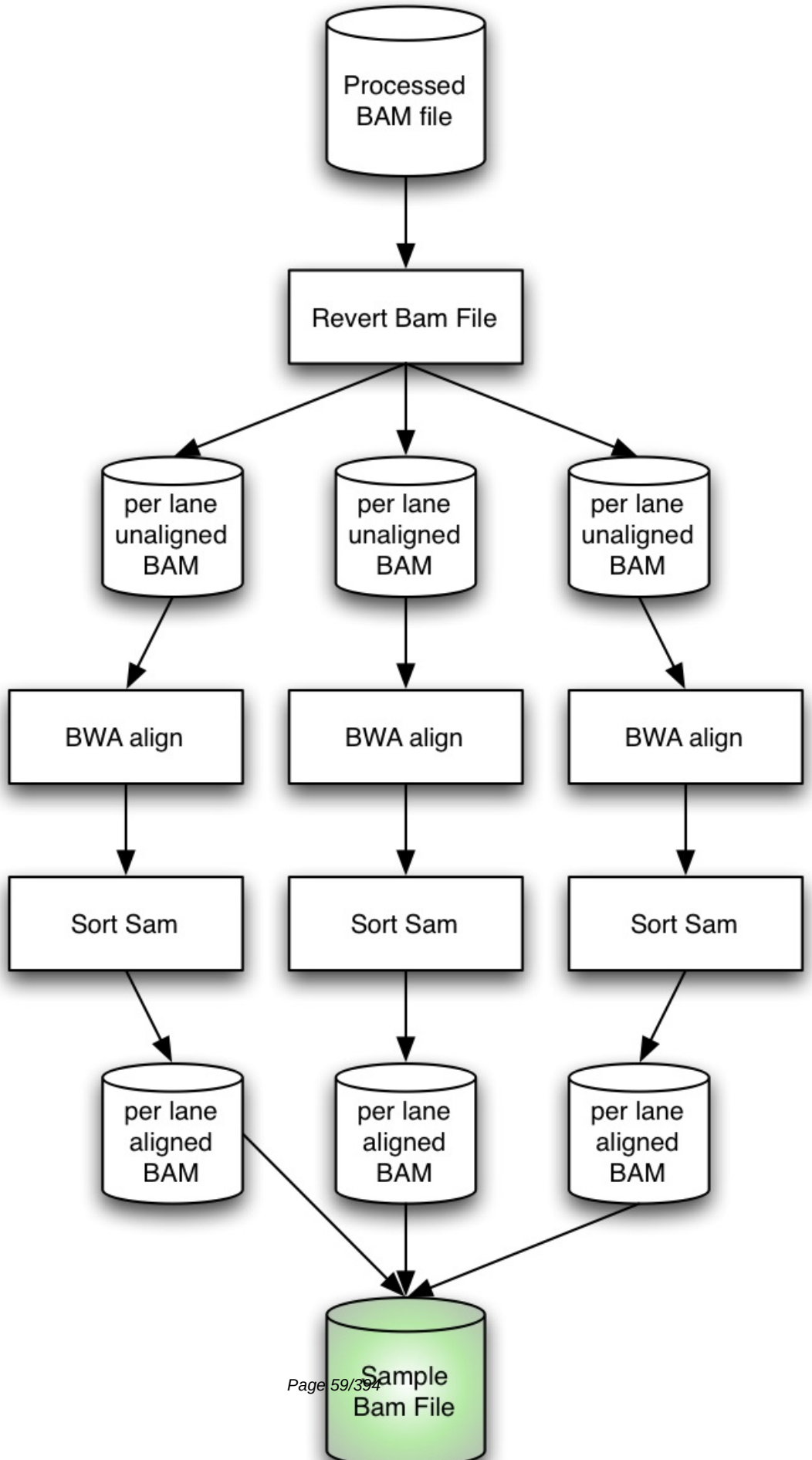
Argument (short-name) Argument (long-name) Description -indels <vcf> --extra_indels <vcf> VCF files to use as reference indels for Indel Realignment. -bwa <path> --path_to_bwa <path> The path to the binary of bwa (usually BAM files have already been mapped - but if you want to remap this is the option) -outputDir < path> --output_directory <path> Output path for the processed BAM files. -L <GATK interval string> --gatk_interval_string <GATK interval string> the -L interval string to be used by GATK - output bams at interval only -intervals <GATK interval file> --gatk_interval_file <GATK interval file> an *intervals* file to be used by GATK - output bams at intervals

Modes of Operation (also optional parameters)

Argument (short-name) Argument (long-name) Description -p <name> --project <name> the project name determines the final output (BAM file) base name. Example NA12878 yields NA12878.processed.bam -knowns --knowns_only Perform cleaning on knowns only. -sw --use_smith_waterman Perform cleaning using Smith Waterman -bwase --use_bwa_single_ended Decompose input BAM file and fully realign it using BWA and assume Single Ended reads -bwape --use_bwa_pair_ended Decompose input BAM file and fully realign it using BWA and assume Pair Ended reads

The Pipeline

Data processing pipeline of the best practices for raw data processing, from sequencer data (fastq files) to analysis read reads (bam file):


Processed BAM file
Revert Bam File
per lane unaligned BAM
per lane unaligned BAM
per lane unaligned BAM
BWA align
BWA align
BWA align
Sort Sam
Sort Sam
Sort Sam
per lane aligned BAM
per lane aligned BAM
per lane aligned BAM
Sample Bam File

Following the group's Best Practices definition, the data processing pipeline does all the processing at the sample level. There are two high-level parts of the pipeline:

BWA alignment

This option is for datasets that have already been processed using a different pipeline or different criteria, and you want to reprocess it using this pipeline. One example is a BAM file that has been processed at the lane level, or did not perform some of the best practices steps of the current pipeline. By using the optional BWA stage of the processing pipeline, your BAM file will be realigned from scratch before creating sample level bams and entering the pipeline.

Sample Level Processing

This is the where the pipeline applies its main procedures: Indel Realignment and Base Quality Score Recalibration.

Indel Realignment

This is a two step process. First we create targets using the Realigner Target Creator (either for knowns only, or including data indels), then we realign the targets using the Indel Realigner (see [Local realignment around indels]) with an optional smith waterman realignment. The Indel Realigner also fixes mate pair information for reads that get realigned.

Base Quality Score Recalibration

This is a crucial step that re-adjusts the quality score using statistics based on several different covariates. In this pipeline we utilize four: Read Group Covariate, Quality Score Covariate, Cycle Covariate, Dinucleotide Covariate

The Outputs

The Data Processing Pipeline produces 3 types of output for each file: a fully processed bam file, a validation report on the input bam and output bam files, a analysis before and after base quality score recalibration. If you look at the pipeline flowchart, the grey boxes indicate processes that generate an output.

Processed Bam File

The final product of the pipeline is one BAM file per sample in the dataset. It also provides one BAM list with all the bams in the dataset. This file is named <project name>.cohort.list, and each sample bam file has the name < project name>.<sample name>.bam. The sample names are extracted from the input BAM headers, and the project name is provided as a parameter to the pipeline.

Validation Files

We validate each unprocessed sample level BAM file and each final processed sample level BAM file. The validation is performed using Picard's ValidateSamFile. Because the parameters of this validation are very strict, we don't enforce that the input BAM has to pass all validation, but we provide the log of the validation as an informative companion to your input. The validation file is named : <project name>.<sample name> .pre.validation and <project name>.<sample name>.post.validation.

Notice that even if your BAM file fails validation, the pipeline can still go through successfully. The validation is a strict report on how your BAM file is looking. Some errors are not critical, but the output files (both pre.validation and post.validation) should give you some input on how to make your dataset better organized in the BAM format.

Base Quality Score Recalibration Analysis

PDF plots of the base qualities are generated before and after recalibration for further analysis on the impact of recalibrating the base quality scores in each sample file. These graphs are explained in detail here. The plots are created in directories named : <project name>.<sample name>.pre and <project name>.<sample name> .post.

Examples

- Example script that runs the data processing pipeline with its standard parameters and uses LSF for scatter/gathering (without bwa)

```
java \
    -Xmx4g \
    -Djava.io.tmpdir=/path/to/tmpdir \
    -jar path/to/GATK/Queue.jar \
    -S path/to/DataProcessingPipeline.scala \
    -p myFancyProjectName \
    -i myDataSet.list \
    -R reference.fasta \
    -D dbSNP.vcf \
    -run
```

- Performing realignment and the full data processing pipeline in one pair-ended bam file

```
java \
    -Xmx4g \
    -Djava.io.tmpdir=/path/to/tmpdir \
    -jar path/to/Queue.jar \
    -S path/to/DataProcessingPipeline.scala \
    -bwa path/to/bwa \
    -i test.bam \
    -R reference.fasta \
    -D dbSNP.vcf \
    -p myProjectWithRealignment \
    -bwape \
    -run
```

Genotype and Validate #61

Last updated on 2013-03-25 19:58:36

Introduction

Genotype and Validate is a tool to asses the quality of a technology dataset for calling SNPs and Indels given a secondary (validation) datasource.

The simplest scenario is when you have a VCF of hand annotated SNPs and Indels, and you want to know how well a particular technology performs calling these snps. With a dataset (BAM file) generated by the technology in test, and the hand annotated VCF, you can run GenotypeAndValidate to asses the accuracy of the calls with the new technology's dataset.

Another option is to validate the calls on a VCF file, using a deep coverage BAM file that you trust the calls on. The GenotypeAndValidate walker will make calls using the reads in the BAM file and take them as truth, then compare to the calls in the VCF file and produce a truth table.

Command-line arguments

Usage of GenotypeAndValidate and its command line arguments are described here.

The VCF Annotations

The annotations can be either true positive (T) or false positive (F). 'T' means it is known to be a true SNP/Indel, while a 'F' means it is known not to be a SNP/Indel but the technology used to create the VCF calls it. To annotate the VCF, simply add an INFO field GV with the value T or F.

The Outputs

GenotypeAndValidate has two outputs. The truth table and the optional VCF file. The truth table is a 2x2 table correlating what was called in the dataset with the truth of the call (whether it's a true positive or a false positive). The table should look like this:

ALT REF Predictive Value **called alt** True Positive (TP) False Positive (FP) Positive PV **called ref** False Negative (FN) True Negative (TN) Negative PV

The **positive predictive value** (PPV) is the proportion of subjects with positive test results who are correctly diagnose.

The **negative predictive value** (NPV) is the proportion of subjects with a negative test result who are correctly diagnosed.

The optional VCF file will contain only the variants that were called or not called, excluding the ones that were uncovered or didn't pass the filters (-depth). This file is useful if you are trying to compare the PPV and NPV of two different technologies on the exact same sites (so you can compare apples to apples).

Additional Details

- You should always use -BTI alleles, so that the GATK only looks at the sites on the VCF file, speeds up the

process a lot. (this will soon be added as a default gatk engine mode)

- The total number of visited bases may be greater than the number of variants in the original VCF file because of extended indels, as they trigger one call per new insertion or deletion. (i.e. ACTG/- will count as 4 genotyper calls, but it's only one line in the VCF).

Examples

Genotypes BAM file from new technology using the VCF as a truth dataset:

```
java \
     -jar /GenomeAnalysisTK.jar \
     -T  GenotypeAndValidate \
     -R human_g1k_v37.fasta \
     -I myNewTechReads.bam \
     -alleles handAnnotatedVCF.vcf \
     -BTI alleles \
     -o gav.vcf
```

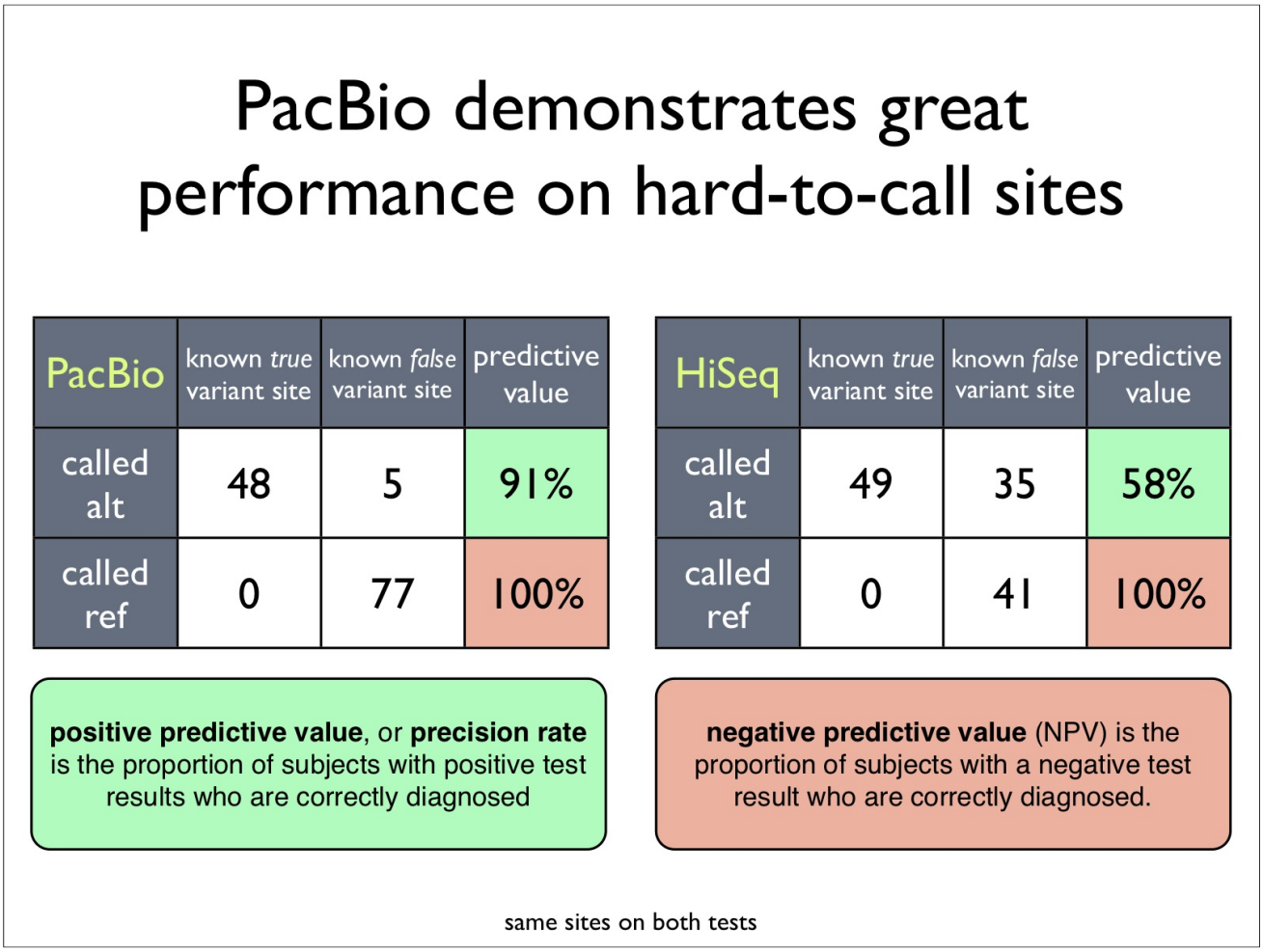
An annotated VCF example (info field clipped for clarity)

```
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  NA12878
1   20568807    .   C   T   0    HapMapHet        AC=1;AF=0.50;AN=2;DP=0;GV=T  GT  0/1
1   22359922    .   T   C   282  WG-CG-HiSeq      AC=2;AF=0.50;GV=T;AN=4;DP=42
GT:AD:DP:GL:GQ  1/0 ./. 0/1:20,22:39:-72.79,-11.75,-67.94:99    ./.
13  102391461   .   G   A   341  Indel;SnpCluster AC=1;GV=F;AF=0.50;AN=2;DP=45
GT:AD:DP:GL:GQ  ./. ./. 0/1:32,13:45:-50.99,-13.56,-112.17:99   ./.
1   175516757   .   C   G   655  SnpCluster,WG    AC=1;AF=0.50;AN=2;GV=F;DP=74
GT:AD:DP:GL:GQ  ./. ./. 0/1:52,22:67:-89.02,-20.20,-191.27:99   ./.
```

Using a BAM file as the truth dataset:

```
java \
     -jar /GenomeAnalysisTK.jar \
     -T  GenotypeAndValidate \
     -R human_g1k_v37.fasta \
     -I myTruthDataset.bam \
     -alleles callsToValidate.vcf \
     -BTI alleles \
     -bt \
     -o gav.vcf
```

Example truth table of PacBio reads (BAM) to validate HiSeq annotated dataset (VCF) using the GenotypeAndValidate walker:



HLA Caller #65

Last updated on 2013-03-25 21:20:43

WARNING: unfortunately we do not have the resources to directly support the HLA typer at this time. As such this tool is no longer under active development or supported by our group. The source code is available in the GATK as is. This tool may or may not work without substantial experimentation by an analyst.

Introduction

Inherited DNA sequence variation in the major histocompatibilty complex (MHC) on human chromosome 6 significantly influences the inherited risk for autoimmune diseases and the host response to pathogenic infections. Collecting allelic sequence information at the classical human leukocyte antigen (HLA) genes is critical for matching in organ transplantation and for genetic association studies, but is complicated due to the high degree of polymorphism across the MHC. Next-generation sequencing offers a cost-effective alternative to Sanger-based sequencing, which has been the standard for classical HLA typing. To bridge the gap between

traditional typing and newer sequencing technologies, we developed a generic algorithm to call HLA alleles at 4-digit resolution from next-generation sequence data.

Downloading the HLA tools

The HLA-specific walkers/tools (FindClosestHLA, CalculateBaseLikelihoods, and HLACaller) are available as a separate download from our FTP site and as source code only. Instructions for obtaining and compiling them are as follows:

Download the source code (in a tar ball):

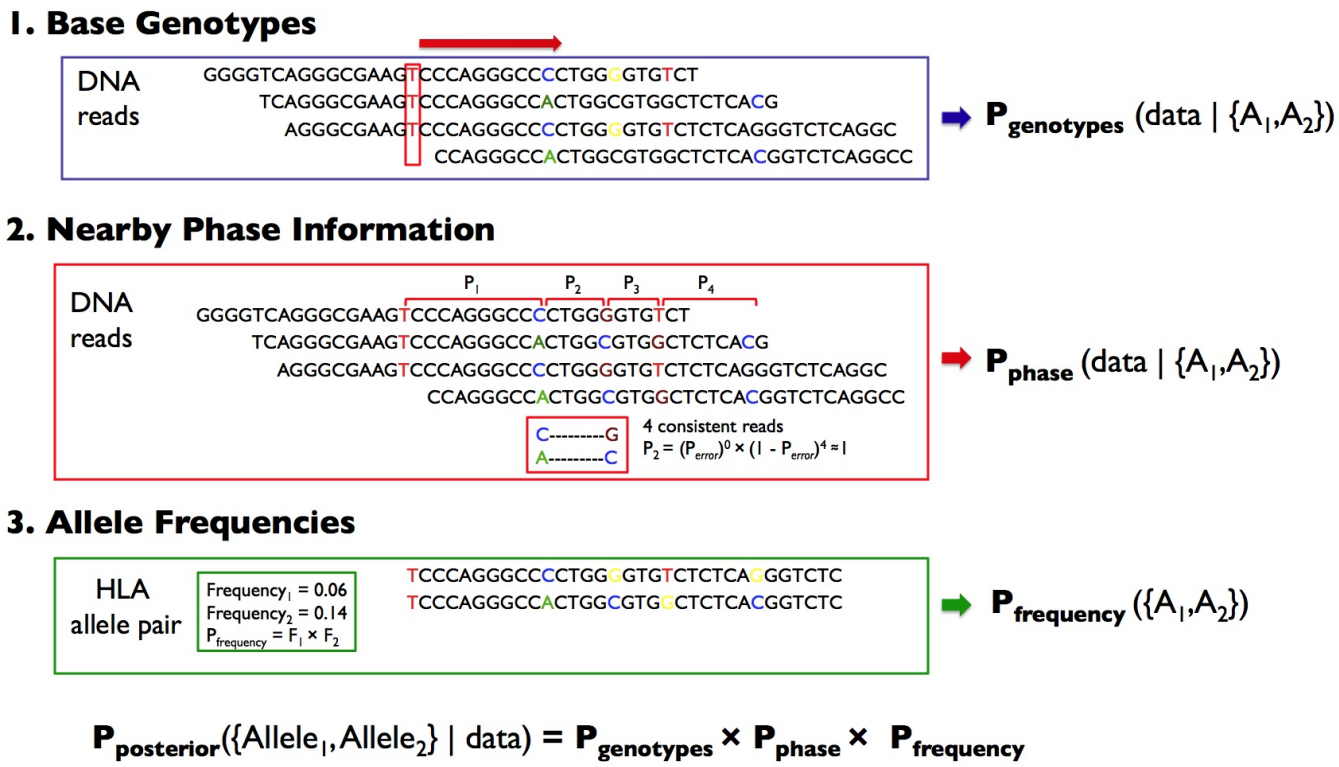
location: ftp://gsapubftp-anonymous@ftp.broadinstitute.org password: <blank> subdirectory: HLA/

Untar the file, 'cd' into the untar'ed directory and compile with 'ant'

Remember that we no longer support this tool, so if you encounter issues with any of these steps please do NOT post them to our support forum.

The algorithm

Algorithmic components of the HLA caller:



The HLA caller algorithm, developed as part of the open-source GATK, examines sequence reads aligned to the classical HLA loci taking SAM/BAM formatted files as input and calculates, for each locus, the posterior

probabilities for all pairs of classical alleles based on three key considerations: (1) genotype calls at each base position, (2) phase information of nearby variants, and (3) population-specific allele frequencies. See the diagram below for a visualization of the heuristic. The output of the algorithm is a list of HLA allele pairs with the highest posterior probabilities.

Functionally, the HLA caller was designed to run in three steps:

- The "FindClosestAllele" walker detects misaligned reads by comparing each read to the dictionary of HLA alleles (reads with < 75% SNP homology to the closest matching allele are removed)

- The "CalculateBaseLikelihoods" walker calculates the likelihoods for each genotype at each position within the HLA loci and finds the polymorphic sites in relation to the reference

- The "HLAcaller" walker reads the output of the previous steps, and makes the likelihood / probability calculations based on base genotypes, phase information, and allele frequencies.

Required inputs

These inputs are required:

- Aligned sequence (.bam) file - input data
- Genomic reference (.bam) file - human genome build 36.
- HLA exons (HLA.intervals) file - list of HLA loci / exons to examine.
- HLA dictionary - list of HLA alleles, DNA sequences, and genomic positions.
- HLA allele frequencies - allele frequencies for HLA alleles across multiple populations.
- HLA polymorphic sites - list of polymorphic sites (used by FindClosestHLA walker)

You can download the latter four here.

Usage and Arguments

Standard GATK arguments (applies to subsequent functions)

The GATK contains a wealth of tools for analysis of sequencing data. Required inputs include an aligned bam file and reference fasta file. The following example shows how to calculate depth of coverage.

Usage:

```
java -jar GenomeAnalysisTK.jar -T DepthOfCoverage -I input.bam -R ref.fasta -L
input.intervals > output.doc
```

Arguments:

- `-T` (required) name of walker/function
- `-I` (required) Input (.bam) file.
- `-R` (required) Genomic reference (.fasta) file.
- `-L` (optional) Interval or list of genomic intervals to run the genotyper on.

FindClosestHLA

The FindClosestHLA walker traverses each read and compares it to all overlapping HLA alleles (at specific polymorphic sites), and identifies the closest matching alleles. This is useful for detecting misalignments (low concordance with best-matching alleles), and helps narrow the list of candidate alleles (narrowing the search space reduces computational speed) for subsequent analysis by the HLACaller walker. Inputs include the HLA dictionary, a list of polymorphic sites in the HLA, and the exons of interest. Output is a file (output.filter) that includes the closest matching alleles and statistics for each read.

Usage:

```
java -jar GenomeAnalysisTK.jar -T FindClosestHLA -I input.bam -R ref.fasta -L
HLA_EXONS.intervals -HLAdictionary HLA_DICTIONARY.txt \
     -PolymorphicSites HLA_POLYMORPHIC_SITES.txt -useInterval HLA_EXONS.intervals | grep -v
INFO > output.filter
```

Arguments:

- `-HLAdictionary` (required) HLA_DICTIONARY.txt file
- `-PolymorphicSites` (required) HLA_POLYMORPHIC_SITES.txt file
- `-useInterval` (required) HLA_EXONS.intervals file

CalculateBaseLikelihoods

CalculateBestLikelihoods walker traverses each base position to determine the likelihood for each of the 10 diploid genotypes. These calculations are used later by HLACaller to determine likelihoods for HLA allele pairs based on genotypes, as well as determining the polymorphic sites used in the phasing algorithm. Inputs include aligned bam input, (optional) results from FindClosestHLA (to remove misalignments), and cutoff values for inclusion or exclusion of specific reads. Output is a file (output.baselikelihoods) that contains base likelihoods at each position.

Usage:

```
java -jar GenomeAnalysisTK.jar -T CalculateBaseLikelihoods -I input.bam -R ref.fasta -L
HLA_EXONS.intervals -filter output.filter \
     -maxAllowedMismatches 6 -minRequiredMatches 0  | grep -v "INFO"  | grep -v "MISALIGNED"
> output.baselikelihoods
```

Arguments:

- `-filter` (optional) file = output of FindClosestHLA walker (output.filter - to exclude misaligned reads in genotype calculations)
- `-maxAllowedMismatches` (optional) max number of mismatches tolerated between a read and the closest allele (default = 6)
- `-minRequiredMatches` (optional) min number of base matches required between a read and the closest

allele (default = 0)

HLACaller

The HLACaller walker calculates the likelihoods for observing pairs of HLA alleles given the data based on genotype, phasing, and allele frequency information. It traverses through each read as part of the phasing algorithm to determine likelihoods based on phase information. The inputs include an aligned bam files, the outputs from FindClosestHLA and CalculateBaseLikelihoods, the HLA dictionary and allele frequencies, and optional cutoffs for excluding specific reads due to misalignment (maxAllowedMismatches and minRequiredMatches).

Usage:

```
java -jar GenomeAnalysisTK.jar -T HLACaller -I input.bam -R ref.fasta -L HLA_EXONS.intervals
-filter output.filter -baselikelihoods output.baselikelihoods\
     -maxAllowedMismatches 6 -minRequiredMatches 5  -HLAdictionary HLA_DICTIONARY.txt
-HLAfrequencies HLA_FREQUENCIES.txt | grep -v "INFO"  > output.calls
```

Arguments:

- `-baseLikelihoods` (required) output of CalculateBaseLikelihoods walker (output.baselikelihoods - genotype likelihoods / list of polymorphic sites from the data)
- `-HLAdictionary` (required) HLA_DICTIONARY.txt file
- `-HLAfrequencies` (required) HLA_FREQUENCIES.txt file
- `-useInterval` (required) HLA_EXONS.intervals file
- `-filter` (optional) file = output of FindClosestAllele walker (to exclude misaligned reads in genotype calculations)
- `-maxAllowedMismatches` (optional) max number of mismatched bases tolerated between a read and the closest allele (default = 6)
- `-minRequiredMatches` (optional) min number of base matches required between a read and the closest allele (default = 5)
- `-minFreq` (option) minimum allele frequency required to consider the HLA allele (default = 0.0).

Example workflow (genome-wide HiSeq data in NA12878 from HapMap; computations were performed on the Broad servers)

Extract sequences from the HLA loci and make a new bam file:

```
use Java-1.6

set HLA=/seq/NKseq/sjia/HLA_CALLER
set GATK=/seq/NKseq/sjia/Sting/dist/GenomeAnalysisTK.jar
set REF=/humgen/1kg/reference/human_b36_both.fasta

cp $HLA/samheader NA12878.HLA.sam
```

```
java -jar $GATK -T PrintReads \
     -I /seq/dirseq/ftp/NA12878_exome/NA12878.bam -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta \
     -L $HLA/HLA.intervals | grep -v RESULT | sed 's/chr6/6/g' >> NA12878.HLA.sam

/home/radon01/sjia/bin/SamToBam.csh NA12878.HLA
```

Use FindClosestHLA to find closest matching HLA alleles and to detect possible misalignments:

```
java -jar $GATK -T FindClosestHLA -I NA12878.HLA.bam -R $REF -L $HLA/HLA_EXONS.intervals
-useInterval $HLA/HLA_EXONS.intervals \
     -HLAdictionary $HLA/HLA_DICTIONARY.txt -PolymorphicSites $HLA/HLA_POLYMORPHIC_SITES.txt
| grep -v INFO > NA12878.HLA.filter

READ_NAME   START-END       S   %Match  Matches Discord Alleles
20FUKAAXX100202:7:63:8309:75917 30018423-30018523   1.0 1.000   1   0
HLA_A*01010101,HLA_A*01010102N,HLA_A*010102,HLA_A*010103,HLA_A*010104,...
20GAVAAXX100126:3:24:13495:18608    30018441-30018541   1.0 1.000   3   0
HLA_A*0312,HLA_A*110101,HLA_A*110102,HLA_A*110103,HLA_A*110104,...
20FUKAAXX100202:8:44:16857:92134    30018442-30018517   1.0 1.000   1   0
HLA_A*01010101,HLA_A*01010102N,HLA_A*010102,HLA_A*010103,HLA_A*010104,HLA_A*010105,...
20FUKAAXX100202:8:5:4309:85338  30018452-30018552   1.0 1.000   3   0
HLA_A*0312,HLA_A*110101,HLA_A*110102,HLA_A*110103,HLA_A*110104,HLA_A*110105,...
20GAVAAXX100126:3:28:7925:160832    30018453-30018553   1.0 1.000   3   0
HLA_A*0312,HLA_A*110101,HLA_A*110102,HLA_A*110103,HLA_A*110104,HLA_A*110105,...
20FUKAAXX100202:1:2:10539:169258    30018459-30018530   1.0 1.000   1   0
HLA_A*01010101,HLA_A*01010102N,HLA_A*010102,HLA_A*010103,.
20FUKAAXX100202:8:43:18611:44456    30018460-30018560   1.0 1.000   3   0
HLA_A*01010101,HLA_A*01010102N,HLA_A*010102,HLA_A*010103,HLA_A*010104,...
```

Use CalculateBaseLikelihoods to determine genotype likelihoods at every base position:

```
java -jar $GATK -T CalculateBaseLikelihoods -I NA12878.HLA.bam -R $REF -L
$HLA/HLA_EXONS.intervals \
     -filter NA12878.HLA.filter -maxAllowedMismatches 6 -minRequiredMatches 0 | grep -v INFO
 | grep -v MISALIGNED > NA12878.HLA.baselikelihoods

chr:pos     Ref Counts          AA  AC  AG  AT  CC  CG  CT  GG  GT  TT
6:30018513  G   A[0]C[0]T[1]G[39]   -113.58 -113.58 -13.80  -113.29 -113.58 -13.80  -113.29
-3.09   -13.50  -113.11
6:30018514  C   A[0]C[39]T[0]G[0]   -119.91 -13.00  -119.91 -119.91 -2.28   -13.00  -13.00
-119.91 -119.91 -119.91
6:30018515  T   A[0]C[0]T[39]G[0]   -118.21 -118.21 -118.21 -13.04  -118.21 -118.21 -13.04
-118.21 -13.04  -2.35
6:30018516  C   A[0]C[38]T[1]G[0]   -106.91 -13.44  -106.91 -106.77 -3.05   -13.44  -13.30
-106.91 -106.77 -106.66
```

```
6:30018517  C   A[0]C[38]T[0]G[0]   -103.13 -13.45  -103.13 -103.13 -3.64   -13.45  -13.45
-103.13 -103.13 -103.13
6:30018518  C   A[0]C[38]T[0]G[0]   -112.23 -12.93  -112.23 -112.23 -2.71   -12.93  -12.93
-112.23 -112.23 -112.23
...
```

Run HLACaller using outputs from previous steps to determine the most likely alleles at each locus:

```
java -jar $GATK -T HLACaller -I NA12878.HLA.bam -R $REF -L $HLA/HLA_EXONS.intervals
-useInterval $HLA/HLA_EXONS.intervals \
     -bl NA12878.HLA.baselikelihoods -filter NA12878.HLA.filter -maxAllowedMismatches 6
-minRequiredMatches 5 \
     -HLAdictionary $HLA/HLA_DICTIONARY.txt -HLAfrequencies $HLA/HLA_FREQUENCIES.txt >
NA12878.HLA.info

grep -v INFO NA12878.HLA.info > NA12878.HLA.calls
```

```
Locus   A1  A2  Geno    Phase   Frq1    Frq2    L   Prob    Reads1  Reads2  Locus   EXP
White   Black   Asian
A   0101    1101    -1229.5 -15.2   -0.82   -0.73   -1244.7 1.00    180 191 229 1.62
-1.99   -3.13   -2.07
B   0801    5601    -832.3  -37.3   -1.01   -2.15   -872.1  1.00    58  59  100 1.17
-3.31   -4.10   -3.95
C   0102    0701    -1344.8 -37.5   -0.87   -0.86   -1384.2 1.00    91  139 228 1.01
-2.35   -2.95   -2.31
DPA1    0103    0201    -842.1  -1.8    -0.12   -0.79   -846.7  1.00    72  48  120 1.00
-0.90   -INF    -1.27
DPB1    0401    1401    -991.5  -18.4   -0.45   -1.55   -1010.7 1.00    64  48  113 0.99
-2.24   -3.14   -2.64
DQA1    0101    0501    -1077.5 -15.9   -0.90   -0.62   -1095.4 1.00    160 77  247 0.96
-1.53   -1.60   -1.87
DQB1    0201    0501    -709.6  -18.6   -0.77   -0.76   -729.7  0.95    50  87  137 1.00
-1.76   -1.54   -2.23
DRB1    0101    0301    -1513.8 -317.3  -1.06   -0.94   -1832.6 1.00    52  32  101 0.83
-1.99   -2.83   -2.34
```

Make a SAM/BAM file of the called alleles:

```
awk '{if (NR > 1){print $1 "*" $2 "\n" $1 "*" $3}}' NA12878.HLA.calls | sort -u >
NA12878.HLA.calls.unique
cp $HLA/samheader NA12878.HLA.calls.sam
awk '{split($1,a,"*"); print "grep \"" a[1] "[*]" a[2] "\" '$HLA/HLA_DICTIONARY.sam' >>
'NA12878.HLA'.tmp";}' NA12878.HLA.calls.unique | sh
sort -k4 -n NA12878.HLA.tmp >> NA12878.HLA.calls.sam
/home/radon01/sjia/bin/SamToBam.csh NA12878.HLA.calls
rm NA12878.HLA.tmp
```

Performance Considerations / Tradeoffs

There exist a few performance / accuracy tradeoffs in the HLA caller, as in any algorithm. The following are a few key considerations that the user should keep in mind when using the software for HLA typing.

Robustness to sequencing/alignment artifact vs. Ability to recognize rare alleles

In polymorphic regions of the genome like the HLA, misaligned reads (presence of erroneous reads or lack of proper sequences) and sequencing errors (indels, systematic PCR errors) may cause the HLA caller to call rare alleles with polymorphisms at the affected bases. The user can manually spot these errors when the algorithm calls a rare allele (Frq1 and Frq2 columns in the output of HLACaller indicate log10 of the allele frequencies). Alternatively, the user can choose to consider only non-rare alleles (use the "-minFreq 0.001" option in HLACaller) to make the algorithm (faster and) more robust against sequencing or alignment errors. The drawback to this approach is that the algorithm may not be able to correctly identifying rare alleles when they are truly present. We recommend using the -minFreq option for genome-wide sequencing datasets, but not for high-quality (targeted PCR 454) data specifically captured for HLA typing in large cohorts.

Misalignment Detection and Data Pre-Processing

The FindClosestAllele walker (optional step) is recommended for two reasons:

- The ability to detect misalignments for reads that don't match very well to the closest appearing HLA allele - removing these misaligned reads improves calling accuracy.

- Creating a list of closest-matching HLA alleles reduces the search space (over 3,000 HLA alleles across the class I and class II loci) that HLACaller has to iterate through, reducing computational burdon.

However, using this pre-processing step is not without costs:

- Any cutoff chosen for %concordance, min base matches, or max base mismatches will not distinguish between correctly aligned and misaligned reads 100% of the time - there is a chance that correctly aligned reads may be removed, and misaligned reads not removed.

- The list of closest-matching alleles in some cases may not contain the true allele if there is sufficient sequencing error, in which case the true allele will not be considered by the HLACaller walker. In our experience, the advantages of using this pre-processing FindClosestAllele walker greatly outweighs the disadvantages, as recommend including it in the pipeline long as the user understands the possible risks of using this function.

Contributions

The HLA caller algorithm was was developed by Xiaoming (Sherman) Jia with generous support of the GATK team (especially Mark Depristo, Eric Banks), and Paul de Bakker.

- xiaomingjia at gmail dot com
- depristo at broadinstitute dot org
- ebanks at broadinstitute dot org
- pdebakker at rics dot bwh dot harvard dot edu

Interface with BEAGLE Software #43

Last updated on 2013-09-26 20:34:32

Note: As of version 4, BEAGLE reads and outputs VCF files directly, and can handle multiallelic sites. We have not yet evaluated what this means for the GATK-BEAGLE interface; it is possible that some of the information provided below is no longer applicable as a result.

Introduction

BEAGLE is a state of the art software package for analysis of large-scale genetic data sets with hundreds of thousands of markers genotyped on thousands of samples. BEAGLE can

- phase genotype data (i.e. infer haplotypes) for unrelated individuals, parent-offspring pairs, and parent-offspring trios.
- infer sporadic missing genotype data.
- impute ungenotyped markers that have been genotyped in a reference panel.
- perform single marker and haplotypic association analysis.
- detect genetic regions that are homozygous-by-descent in an individual or identical-by-descent in pairs of individuals.

The GATK provides an experimental interface to BEAGLE. Currently, the only use cases supported by this interface are a) inferring missing genotype data from call sets (e.g. for lack of coverage in low-pass data), b) Genotype inference for unrelated individuals.

Workflow

The basic workflow for this interface is as follows:

After variants are called and possibly filtered, the GATK walker ProduceBeagleInput will take the resulting VCF as input, and will produce a likelihood file in BEAGLE format.

- User needs to run BEAGLE with this likelihood file specified as input.
- After Beagle runs, user must unzip resulting output files (.gprobs, .phased) containing posterior genotype probabilities and phased haplotypes.
- User can then run GATK walker BeagleOutputToVCF to produce a new VCF with updated data. The new VCF will contain updated genotypes as well as updated annotations.

Producing Beagle input likelihoods file

Before running BEAGLE, we need to first take an input VCF file with genotype likelihoods and produce the BEAGLE likelihoods file using walker ProduceBealgeInput, as described in detail in its documentation page.

For each variant in inputvcf.vcf, ProduceBeagleInput will extract the genotype likelihoods, convert from log to linear space, and produce a BEAGLE input file in Genotype likelihoods file format (see BEAGLE documentation for more details). Essentially, this file is a text file in tabular format, a snippet of which is pasted below:

```
marker    alleleA alleleB NA07056 NA07056 NA07056 NA11892 NA11892 NA11892
20:60251    T        C     10.00   1.26    0.00     9.77   2.45    0.00
20:60321    G        T     10.00   5.01    0.01    10.00   0.31    0.00
20:60467    G        C      9.55   2.40    0.00     9.55   1.20    0.00
```

Note that BEAGLE only supports biallelic sites. Markers can have an arbitrary label, but they need to be in chromosomal order. Sites that are not genotyped in the input VCF (i.e. which are annotated with a "./." string and have no Genotype Likelihood annotation) are assigned a likelihood value of (0.33, 0.33, 0.33).

IMPORTANT: Due to BEAGLE memory restrictions, it's strongly recommended that BEAGLE be run on a separate chromosome-by-chromosome basis. In the current use case, BEAGLE uses RAM in a manner approximately proportional to the number of input markers. After BEAGLE is run and an output VCF is produced as described below, CombineVariants can be used to combine resulting VCF's, using the "-variantMergeOptions UNION" argument.

Running Beagle

We currently only support a subset of BEAGLE functionality - only unphased, unrelated input likelihood data is supported. To run imputation analysis, run for example

```
java -Xmx4000m -jar path_to_beagle/beagle.jar like=path_to_beagle_output/beagle_output
out=myrun
```

Extra BEAGLE arguments can be added as required.

About Beagle memory usage

Empirically, Beagle can run up to about ~800,000 markers with 4 GB of RAM. Larger chromosomes require additional memory.

Processing BEAGLE output files

BEAGLE will produce several output files. The following shell commands unzip the output files in preparation for their being processed, and put them all in the same place:

```
# unzip gzip'd files, force overwrite if existing
gunzip -f path_to_beagle_output/myrun.beagle_output.gprobs.gz
gunzip -f path_to_beagle_output/myrun.beagle_output.phased.gz
#rename also Beagle likelihood file to mantain consistency
mv path_to_beagle_output/beagle_output path_to_beagle_output/myrun.beagle_output.like
```

Creating a new VCF from BEAGLE data with BeagleOutputToVCF

Once BEAGLE files are produced, we can update our original VCF with BEAGLE's data. This is done using the BeagleOutputToVCF tool.

The walker looks for the files specified with the -B(type,BEAGLE,file) triplets as above for the output posterior

genotype probabilities, the output r^2 values and the output phased genotypes. The order in which these are given in the command line is arbitrary, but all three must be present for correct operation.

The output VCF has the new genotypes that Beagle produced, and several annotations are also updated. By default, the walker will update the per-genotype annotations GQ (Genotype Quality), the genotypes themselves, as well as the per-site annotations AF (Allele Frequency), AC (Allele Count) and AN (Allele Number).

The resulting VCF can now be used for further downstream analysis.

Merging VCFs broken up by chromosome into a single genome-wide file

Assuming you have broken up your calls into Beagle by chromosome (as recommended above), you can use the CombineVariants tool to merge the resulting VCFs into a single callset.

```
java -jar /path/to/dist/GenomeAnalysisTK.jar \
     -T CombineVariants \
     -R reffile.fasta \
     --out genome_wide_output.vcf \
     -V:input1 beagle_output_chr1.vcf \
     -V:input2 beagle_output_chr2.vcf \
  .
  .
  .
     -V:inputX beagle_output_chrX.vcf \
     -type UNION -priority input1,input2,...,inputX
```

Lifting over a VCF from one reference to another #63

Last updated on 2013-09-13 19:39:58

liftOverVCF.pl

Contents

- 1 Introduction
- 2 Obtaining the Script
- 3 Example
- 4 Usage
- 5 Chain files

Introduction

This script converts a VCF file from one reference build to another. It runs 3 modules within our toolkit that are necessary for lifting over a VCF.

1. LiftoverVariants walker

2. sortByRef.pl to sort the lifted-over file
3. Filter out records whose ref field no longer matches the new reference

Obtaining the Script

The liftOverVCF.pl script is available in our public source repository under the 'perl' directory. Instructions for pulling down our source are available here.

Example

```
./liftOverVCF.pl -vcf calls.b36.vcf \
     -chain b36ToHg19.broad.over.chain \
     -out calls.hg19.vcf \
     -gatk /humgen/gsa-scr1/ebanks/Sting_dev
     -newRef /seq/references/Homo_sapiens_assembly19/v0/Homo_sapiens_assembly19
     -oldRef /humgen/1kg/reference/human_b36_both
     -tmp /broad/shptmp [defaults to /tmp]
```

Usage

Running the script with no arguments will show the usage:

```
Usage: liftOverVCF.pl
     -vcf        <input vcf>
     -gatk       <path to gatk trunk>
     -chain      <chain file>
     -newRef     <path to new reference prefix; we will need newRef.dict, .fasta, and
.fasta.fai>
     -oldRef     <path to old reference prefix; we will need oldRef.fasta>
     -out        <output vcf>
     -tmp            <temp file location; defaults to /tmp>
```

- The 'tmp' argument is optional. It specifies the location to write the temporary file from step 1 of the process.

Chain files

Chain files from b36/hg18 to hg19 are located here within the Broad:

```
/humgen/gsa-hpprojects/GATK/data/Liftover_Chain_Files/
```

External users can get them off our ftp site:

```
location: ftp.broadinstitute.org
username: gsapubftp-anonymous
path:     Liftover_Chain_Files
```

Local Realignment around Indels #38

Last updated on 2012-09-30 23:35:55

Realigner Target Creator

For a complete, detailed argument reference, refer to the GATK document page here.

Indel Realigner

For a complete, detailed argument reference, refer to the GATK document page here.

Running the Indel Realigner only at known sites

While we advocate for using the Indel Realigner over an aggregated bam using the full Smith-Waterman alignment algorithm, it will work for just a single lane of sequencing data when run in -knownsOnly mode. Novel sites obviously won't be cleaned up, but the majority of a single individual's short indels will already have been seen in dbSNP and/or 1000 Genomes. One would employ the known-only/lane-level realignment strategy in a large-scale project (e.g. 1000 Genomes) where computation time is severely constrained and limited. We modify the example arguments from above to reflect the command-lines necessary for known-only/lane-level cleaning.

The RealignerTargetCreator step would need to be done just once for a single set of indels; so as long as the set of known indels doesn't change, the output.intervals file from below would never need to be recalculated.

```
java -Xmx1g -jar /path/to/GenomeAnalysisTK.jar \
    -T RealignerTargetCreator \
    -R /path/to/reference.fasta \
    -o /path/to/output.intervals \
```

```
     -known /path/to/indel_calls.vcf
```

The IndelRealigner step needs to be run on every bam file.

```
java -Xmx4g -Djava.io.tmpdir=/path/to/tmpdir \
     -jar /path/to/GenomeAnalysisTK.jar \
     -I <lane-level.bam> \
     -R <ref.fasta> \
     -T IndelRealigner \
     -targetIntervals <intervalListFromStep1Above.intervals> \
     -o <realignedBam.bam> \
     -known /path/to/indel_calls.vcf
     --consensusDeterminationModel KNOWNS_ONLY \
     -LOD 0.4
```

Merging batched call sets #46

Last updated on 2013-05-03 14:46:00

Introduction

Three-stage procedure:

- Create a master set of sites from your N batch VCFs that you want to genotype in all samples. At this stage you need to determine how you want to resolve disagreements among the VCFs. This is your master sites VCF.
- Take the master sites VCF and genotype each sample BAM file at these sites
- (Optionally) Merge the single sample VCFs into a master VCF file

Creating the master set of sites: SNPs and Indels

The first step of batch merging is to create a master set of sites that you want to genotype in all samples. To make this problem concrete, suppose I have two VCF files:

Batch 1:

```
##fileformat=VCFv4.0
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  NA12891
20      9999996     .       A       ATC     .       PASS    .       GT:GQ   0/1:30
20      10000000        .       T       G       .       PASS    .       GT:GQ   0/1:30
20      10000117        .       C       T       .       FAIL    .       GT:GQ   0/1:30
20      10000211        .       C       T       .       PASS    .       GT:GQ   0/1:30
20      10001436        .       A       AGG     .       PASS    .       GT:GQ   1/1:30
```

Batch 2:

```
##fileformat=VCFv4.0
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  NA12878
20      9999996     .       A       ATC     .       PASS    .       GT:GQ   0/1:30
20      10000117        .       C       T       .       FAIL    .       GT:GQ   0/1:30
20      10000211        .       C       T       .       FAIL    .       GT:GQ   0/1:30
20      10000598        .       T       A       .       PASS    .       GT:GQ   1/1:30
20      10001436        .       A       AGGCT   .       PASS    .       GT:GQ   1/1:30
```

In order to merge these batches, I need to make a variety of bookkeeping and filtering decisions, as outlined in the merged VCF below:

Master VCF:

```
20      9999996     .       A       ATC     .       PASS    .       GT:GQ   0/1:30  [pass in
both]
20      10000000        .       T       G       .       PASS    .       GT:GQ   0/1:30
[only in batch 1]
20      10000117        .       C       T       .       FAIL    .       GT:GQ   0/1:30
[fail in both]
20      10000211        .       C       T       .       FAIL    .       GT:GQ   0/1:30
[pass in 1, fail in 2, choice in unclear]
20      10000598        .       T       A       .       PASS    .       GT:GQ   1/1:30
[only in batch 2]
20      10001436        .       A       AGGCT   .       PASS    .       GT:GQ   1/1:30
[A/AGG in batch 1, A/AGGCT in batch 2, including this site may be problematic]
```

These issues fall into the following categories:

- For sites present in all VCFs (20:9999996 above), the alleles agree, and each site PASS is pass, this site can obviously be considered "PASS" in the master VCF

- Some sites may be PASS in one batch, but absent in others (20:10000000 and 20:10000598), which occurs when the site is polymorphic in one batch but all samples are reference or no-called in the other batch

- Similarly, sites that are fail in all batches in which they occur can be safely filtered out, or included as failing filters in the master VCF (20:10000117)

There are two difficult situations that must be addressed by the needs of the project merging batches:

- Some sites may be PASS in some batches but FAIL in others. This might indicate that either:

- The site is actually truly polymorphic, but due to limited coverage, poor sequencing, or other issues it is flag as unreliable in some batches. In these cases, it makes sense to include the site

- The site is actually a common machine artifact, but just happened to escape standard filtering in a few batches. In these cases, you would obviously like to filter out the site

- Even more complicated, it is possible that in the PASS batches you have found a reliable allele (C/T, for

example) while in others there's no alt allele but actually a low-frequency error, which is flagged as failing. Ideally, here you could filter out the failing allele from the FAIL batches, and keep the pass ones

- Some sites may have multiple segregating alleles in each batch. Such sites are often errors, but in some cases may be actual multi-allelic sites, in particular for indels.

Unfortunately, we cannot determine which is actually the correct choice, especially given the goals of the project. We leave it up the project bioinformatician to handle these cases when creating the master VCF. We are hopeful that at some point in the future we'll have a consensus approach to handle such merging, but until then this will be a manual process.

The GATK tool CombineVariants can be used to merge multiple VCF files, and parameter choices will allow you to handle some of the above issues. With tools like SelectVariants one can slice-and-dice the merged VCFs to handle these complexities as appropriate for your project's needs. For example, the above master merge can be produced with the following CombineVariants:

```
java -jar dist/GenomeAnalysisTK.jar \
     -T CombineVariants \
     -R human_g1k_v37.fasta \
     -V:one,VCF combine.1.vcf -V:two,VCF combine.2.vcf \
     --sites_only \
     -minimalVCF \
     -o master.vcf
```

producing the following VCF:

```
##fileformat=VCFv4.0
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
20      9999996     .       A       ACT         .       PASS    set=Intersection
20      10000000        .       T       G           .   PASS    set=one
20      10000117        .       C       T           .       FAIL    set=FilteredInAll
20      10000211        .       C       T           .       PASS    set=filterIntwo-one
20      10000598        .       T       A           .       PASS    set=two
20      10001436        .       A       AGG,AGGCT       .       PASS    set=Intersection
```

Genotyping your samples at these sites

Having created the master set of sites to genotype, along with their alleles, as in the previous section, you now use the UnifiedGenotyper to genotype each sample independently at the master set of sites. This GENOTYPE_GIVEN_ALLELES mode of the UnifiedGenotyper will jump into the sample BAM file, and calculate the genotype and genotype likelihoods of the sample at the site for each of the genotypes available for the REF and ALT alleles. For example, for site 10000211, the UnifiedGenotyper would evaluate the likelihoods of the CC, CT, and TT genotypes for the sample at this site, choose the most likely configuration, and generate a VCF record containing the genotype call and the likelihoods for the three genotype configurations.

As a concrete example command line, you can genotype the master.vcf file using in the bundle sample NA12878 with the following command:

```
java -Xmx2g -jar dist/GenomeAnalysisTK.jar \
     -T UnifiedGenotyper \
     -R bundle/b37/human_g1k_v37.fasta \
     -I bundle/b37/NA12878.HiSeq.WGS.bwa.cleaned.recal.hg19.20.bam \
     -alleles master.vcf \
     -L master.vcf \
     -gt_mode GENOTYPE_GIVEN_ALLELES \
     -out_mode EMIT_ALL_SITES \
     -stand_call_conf 0.0 \
     -glm BOTH \
     -G none \
```

The `-L master.vcf` argument tells the UG to only genotype the sites in the master file. If you don't specify this, the UG will genotype the master sites in GGA mode, but it will also genotype all other sites in the genome in regular mode.

The `last item,-G ` prevents the UG from computing annotations you don't need. This command produces something like the following output:

```
##fileformat=VCFv4.0
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  NA12878
20      9999996     .       A       ACT         4576.19 .       .   GT:DP:GQ:PL
1/1:76:99:4576,229,0
20      10000000        .       T       G           0       .       .       GT:DP:GQ:PL
0/0:79:99:0,238,3093
20      10000211        .       C       T       857.79  .       .   GT:AD:DP:GQ:PL
0/1:28,27:55:99:888,0,870
20      10000598        .       T       A           1800.57 .       .   GT:AD:DP:GQ:PL
1/1:0,48:48:99:1834,144,0
20      10001436        .       A       AGG,AGGCT       1921.12 .       .   GT:DP:GQ:PL
0/2:49:84.06:1960,2065,0,2695,222,84
```

Several things should be noted here:

- The genotype likelihoods calculation evolves, especially for indels, the exact results of this command will change.
- The command will emit sites that are hom-ref in the sample at the site, but the -stand_call_conf 0.0 argument should be provided so that they aren't tagged as "LowQual" by the UnifiedGenotyper.
- The filtered site 10000117 in the master.vcf is not genotyped by the UG, as it doesn't pass filters and so is considered bad by the GATK UG. If you want to determine the genotypes for all sites, independent on filtering, you must unfilter all of your records in master.vcf, and if desired, restore the filter string for these records later.

This genotyping command can be performed independently per sample, and so can be parallelized easily on a farm with one job per sample, as in the following:

```
foreach sample in samples:
  run UnifiedGenotyper command above with -I $sample.bam -o $sample.vcf
end
```

(Optional) Merging the sample VCFs together

You can use a similar command for CombineVariants above to merge back together all of your single sample genotyping runs. Suppose all of my UnifiedGenotyper jobs have completed, and I have VCF files named sample1.vcf, sample2.vcf, to sampleN.vcf. The single command:

```
java -jar dist/GenomeAnalysisTK.jar -T CombineVariants -R human_g1k_v37.fasta -V:sample1
sample1.vcf -V:sample2 sample2.vcf [repeat until] -V:sampleN sampleN.vcf -o combined.vcf
```

General notes

- Because the GATK uses dynamic downsampling of reads, it is possible for truly marginal calls to change likelihoods from discovery (processing the BAM incrementally) vs. genotyping (jumping into the BAM). Consequently, do not be surprised to see minor differences in the genotypes for samples from discovery and genotyping.

- More advanced users may want to consider group several samples together for genotyping. For example, 100 samples could be genotyped in 10 groups of 10 samples, resulting in only 10 VCF files. Merging the 10 VCF files may be faster (or just easier to manage) than 1000 individual VCFs.

- Sometimes, using this method, a monomorphic site within a batch will be identified as polymorphic in one or more samples within that same batch. This is because the UnifiedGenotyper applies a frequency prior to determine whether a site is likely to be monomorphic. If the site is monomorphic, it is either not output, or if EMIT_ALL_SITES is thrown, reference genotypes are output. If the site is determined to be polymorphic, genotypes are assigned greedily (as of GATK-v1.4). Calling single-sample reduces the effect of the prior, so sites which were considered monomorphic within a batch could be considered polymorphic within a sub-batch.

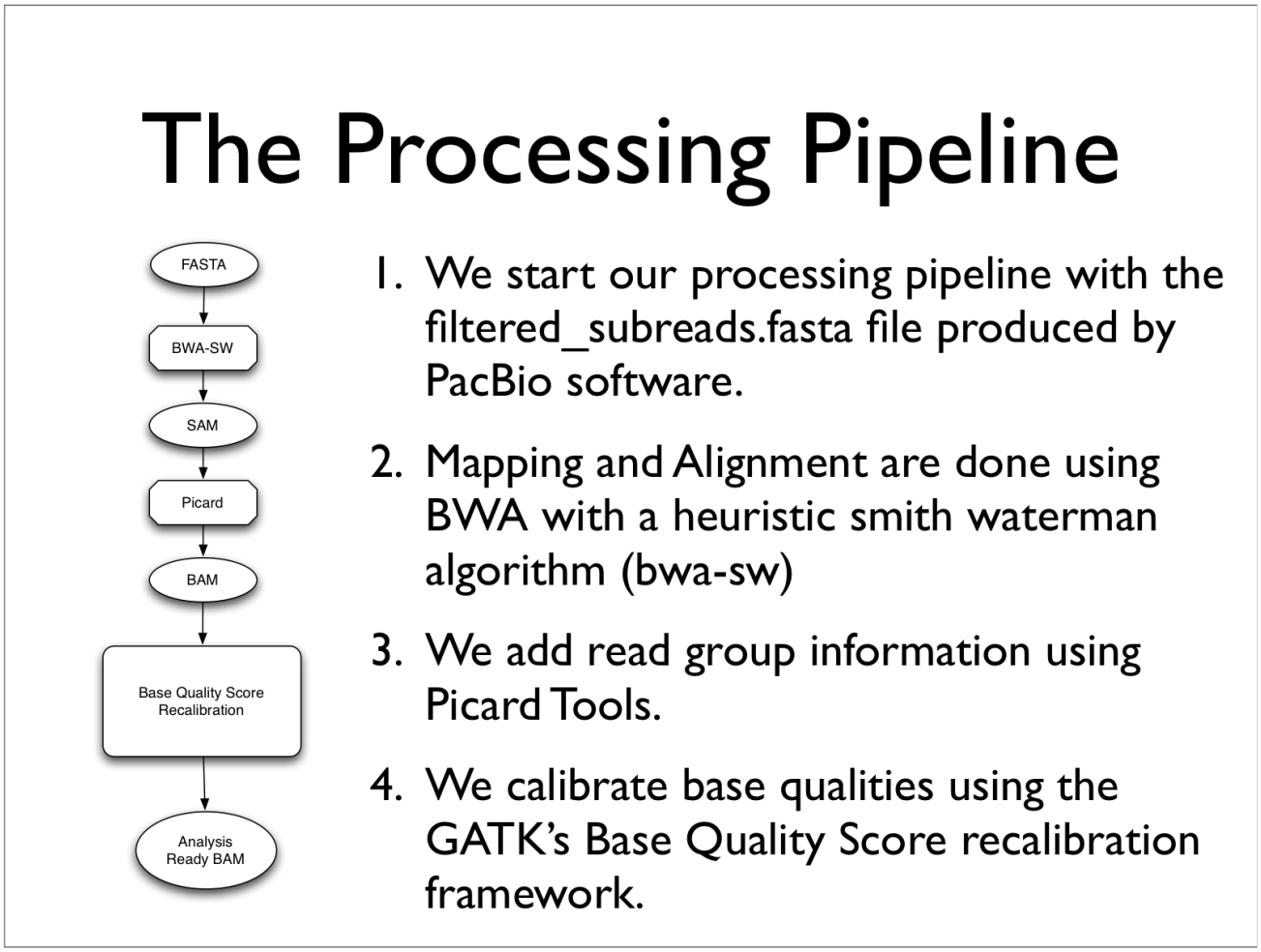
PacBio Data Processing Guidelines #42

Last updated on 2013-03-25 22:18:53

Introduction

Processing data originated in the Pacific Biosciences RS platform has been evaluated by the GSA and publicly presented in numerous occasions. The guidelines we describe in this document were the result of a systematic technology development experiment on some datasets (human, E. coli and Rhodobacter) from the Broad Institute. These guidelines produced better results than the ones obtained using alternative pipelines up to this date (september 2011) for the datasets tested, **but there is no guarantee that it will be the best for every dataset and that other pipelines won't supersede it in the future**.

The pipeline we propose here is illustrated in a Q script (PacbioProcessingPipeline.scala) distributed with the GATK as an example for educational purposes. This pipeline has not been extensively tested and is not supported by the GATK team. You are free to use it and modify it for your needs following the guidelines below.



BWA alignment

First we take the filtered_subreads.fq file output by the Pacific Biosciences RS SMRT pipeline and align it using BWA. We use BWA with the `bwasw` algorithm and allow for relaxing the gap open penalty to account for the excess of insertions and deletions known to be typical error modes of the data. For an idea on what parameters to use check suggestions given by the BWA author in the BWA manual page that are specific to Pacbio. The goal is to account for Pacific Biosciences RS known error mode and benefit from the long reads for a high scoring overall match. (for older versions, you can use the filtered_subreads.fasta and combine the base quality scores extracted from the h5 files using Pacific Biosciences SMRT pipeline python tools)

To produce a BAM file that is sorted by coordinate with adequate read group information we use Picard tools: SortSam and AddOrReplaceReadGroups. These steps are necessary because all subsequent tools require that the BAM file follow these rules. It is also generally considered good practices to have your BAM file conform to these specifications.

Best Practices for Variant Calling

Once we have a proper BAM file, it is important to estimate the empirical quality scores using statistics based on a known callset (e.g. latest dbSNP) and the following covariates: QualityScore, Dinucleotide and ReadGroup. You can follow the GATK's Best Practices for Variant Detection according the type of data you have, with the exception of indel realignment, because the tool has not been adapted for Pacific Biosciences RS data.

Problems with Variant Calling with Pacific Biosciences

- Calling must be more permissive of indels in the data.

You will have to adjust your calling thresholds in the Unified Genotyper to allow sites with a higher indel rate to be analyzed.

- Base quality thresholds should be adjusted to the specifics of your data.

Be aware that the Unified Genotyper has cutoffs for base quality score and if your data is on average Q20 (a common occurrence with Pacific Biosciences RS data) you may need to adjust your quality thresholds to allow the GATK to analyze your data. There is no right answer here, you have to choose parameters consistent with your average base quality scores, evaluate the calls made with the selected threshold and modify as necessary.

- Reference bias

To account for the high insertion and deletion error rate of the Pacific Biosciences data instrument, we often have to set the gap open penalty to be lower than the base mismatch penalty in order to maximize alignment performance. Despite aligning most of the reads successfully, this creates the side effect that the aligner will sometimes prefer to "hide" a true SNP inside an insertion. The result is accurate mapping, albeit with a reference-biased alignment. It is important to note however, that reference bias is an artifact of the alignment process, not the data, and can be greatly reduced by locally realigning the reads based on the reference and the data. Presently, the available software for local realignment is not compatible with the length and the high indel rate of Pacific Bioscience data, but we expect new tools to handle this problem in the future. Ultimately reference bias will mask real calls and you will have to inspect these by hand.

Pedigree Analysis #37

Last updated on 2013-05-07 17:27:19

Workflow

To call variants with the GATK using pedigree information, you should base your workflow on the Best Practices recommendations -- the principles detailed there all apply to pedigree analysis.

But there is one crucial addition: you should make sure to pass a pedigree file (PED file) to all GATK walkers that you use in your workflow. Some will deliver better results if they see the pedigree data.

At the moment there are two of the standard annotations affected by pedigree:

- Allele Frequency (computed on founders only)

- Inbreeding coefficient (computed on founders only)

Trio Analysis

In the specific case of trios, an additional GATK walker, PhaseByTransmission, should be used to obtain trio-aware genotypes as well as phase by descent.

Important note

The annotations mentioned above have been adapted for PED files starting with GATK v.1.6. If you already have VCF files generated by an older version of the GATK or have not passed a PED file while running the UnifiedGenotyper or VariantAnnotator, you should do the following:

- Run the latest version of the VariantAnnotator to re-annotate your variants.
- Re-annotate all the standard annotations by passing the argument `-G StandardAnnotation` to VariantAnnotator. Make sure you pass your PED file to the VariantAnnotator as well!
- If you are using Variant Quality Score Recalibration (VQSR) with the InbreedingCoefficient as an annotation in your model, you should re-run VQSR once the InbreedingCoefficient is updated.

PED files

The PED files used as input for these tools are based on PLINK pedigree files. The general description can be found here.

For these tools, the PED files must contain only the first 6 columns from the PLINK format PED file, and no alleles, like a FAM file in PLINK.

Per-base alignment qualities (BAQ) in the GATK #1326

Last updated on 2012-10-18 15:33:27

1. Introduction

The GATK provides an implementation of the Per-Base Alignment Qualities (BAQ) developed by Heng Li in late 2010. See this SamTools page for more details.

2. Using BAQ

The BAQ algorithm is applied by the GATK engine itself, which means that all GATK walkers can potentially benefit from it. By default, BAQ is OFF, meaning that the engine will not use BAQ quality scores at all.

The GATK engine accepts the argument `-baq` with the following `enum` values:

```
public enum CalculationMode {
    OFF,                        // don't apply a BAQ at all, the default
    CALCULATE_AS_NECESSARY,     // do HMM BAQ calculation on the fly, as necessary, if
there's no tag
    RECALCULATE                 // do HMM BAQ calculation on the fly, regardless of whether
there's a tag present
```

```
}
```

If you want to enable BAQ, the usual thing to do is CALCULATE_AS_NECESSARY, which will calculate BAQ values if they are not in the BQ read tag. If your reads are already tagged with BQ values, then the GATK will use those. RECALCULATE will always recalculate the BAQ, regardless of the tag, which is useful if you are experimenting with the gap open penalty (see below).

If you are really an expert, the GATK allows you to specify the BAQ gap open penalty (-baqGOP) to use in the HMM. This value should be 40 by default, a good value for whole genomes and exomes for highly sensitive calls. However, if you are analyzing exome data only, you may want to use 30, which seems to result in more specific call set. We continue to play with these values some. Some walkers, where BAQ would corrupt their analyses, forbid the use of BAQ and will throw an exception if -baq is provided.

3. Some example uses of the BAQ in the GATK

- For UnifiedGenotyper to get more specific SNP calls.
- For PrintReads to write out a BAM file with BAQ tagged reads
- For TableRecalibrator or IndelRealigner to write out a BAM file with BAQ tagged reads. Make sure you use -baq RECALCULATE so the engine knows to recalculate the BAQ after these tools have updated the base quality scores or the read alignments. Note that both of these tools will not use the BAQ values on input, but will write out the tags for analysis tools that will use them.

Note that some tools should not have BAQ applied to them.

This last option will be a particularly useful for people who are already doing base quality score recalibration. Suppose I have a pipeline that does:

```
RealignerTargetCreator
IndelRealigner

CountCovariates
TableRecalibrate

UnifiedGenotyper
```

A highly efficient BAQ extended pipeline would look like

```
RealignerTargetCreator
IndelRealigner // don't bother with BAQ here, since we will calculate it in table
recalibrator

CountCovariates
TableRecalibrate -baq RECALCULATE // now the reads will have a BAQ tag added.  Slows the
tool down some

UnifiedGenotyper -baq CALCULATE_AS_NECESSARY // UG will use the tags from TableRecalibrate,
```

```
keeping UG fast
```

4. BAQ and walker control

Walkers can control via the `@BAQMode` annotation how the BAQ calculation is applied. Can either be as a tag, by overwriting the qualities scores, or by only returning the baq-capped qualities scores. Additionally, walkers can be set up to have the BAQ applied to the incoming reads (`ON_INPUT`, the default), to output reads (`ON_OUTPUT`), or `HANDLED_BY_WALKER`, which means that calling into the BAQ system is the responsibility of the individual walker.

Read-backed Phasing #45

Last updated on 2012-09-28 17:42:42

Read-backed Phasing

Example and Command Line Arguments

For a complete, detailed argument reference, refer to the GATK document page here

Introduction

The biological unit of inheritance from each parent in a diploid organism is a set of single chromosomes, so that a diploid organism contains a set of pairs of corresponding chromosomes. The full sequence of each inherited chromosome is also known as a haplotype. It is critical to ascertain which variants are associated with one another in a particular individual. For example, if an individual's DNA possesses two consecutive heterozygous sites in a protein-coding sequence, there are two alternative scenarios of how these variants interact and affect the phenotype of the individual. In one scenario, they are on two different chromosomes, so each one has its own separate effect. On the other hand, if they co-occur on the same chromosome, they are thus expressed in the same protein molecule; moreover, if they are within the same codon, they are highly likely to encode an amino acid that is non-synonymous (relative to the other chromosome). The ReadBackedPhasing program serves to discover these haplotypes based on high-throughput sequencing reads.

The first step in phasing is to call variants ("genotype calling") using a SAM/BAM file of reads aligned to the reference genome -- this results in a VCF file. Using the VCF file and the SAM/BAM reads file, the ReadBackedPhasing tool considers all reads within a Bayesian framework and attempts to find the local haplotype with the highest probability, based on the reads observed.

The local haplotype and its phasing is encoded in the VCF file as a "|" symbol (which indicates that the alleles of the genotype correspond to the same order as the alleles for the genotype at the preceding variant site). For example, the following VCF indicates that SAMP1 is heterozygous at chromosome 20 positions 332341 and 332503, and the reference base at the first position (A) is on the same chromosome of SAMP1 as the alternate base at the latter position on that chromosome (G), and vice versa (G with C):

```
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  SAMP1
```

```
chr20   332341  rs6076509   A   G   470.60  PASS
AB=0.46;AC=1;AF=0.50;AN=2;DB;DP=52;Dels=0.00;HRun=1;HaplotypeScore=0.98;MQ=59.11;MQ0=0;OQ=62
7.69;QD=12.07;SB=-145.57    GT:DP:GL:GQ 0/1:46:-79.92,-13.87,-84.22:99
chr20   332503  rs6133033   C   G   726.23  PASS
AB=0.57;AC=1;AF=0.50;AN=2;DB;DP=61;Dels=0.00;HRun=1;HaplotypeScore=0.95;MQ=60.00;MQ0=0;OQ=89
4.70;QD=14.67;SB=-472.75    GT:DP:GL:GQ:PQ  1|0:60:-110.83,-18.08,-149.73:99:126.93
```

The per-sample per-genotype PQ field is used to provide a Phred-scaled phasing quality score based on the statistical Bayesian framework employed for phasing. Note that for cases of homozygous sites that lie in between phased heterozygous sites, these homozygous sites will be phased with the same quality as the next heterozygous site.

Limitations:

- ReadBackedPhasing doesn't currently support insertions, deletions, or multi-nucleotide polymorphisms.
- Input VCF files should only be for diploid organisms.

More detailed aspects of semantics of phasing in the VCF format

- The "|" symbol is used for each sample to indicate that each of the alleles of the genotype in question derive from the same haplotype as each of the alleles of the genotype of the same sample in the previous **NON-FILTERED** variant record. That is, rows without FILTER=PASS are essentially ignored in the read-backed phasing (RBP) algorithm.
- Note that the first heterozygous genotype record in a pair of haplotypes will necessarily have a "/" - otherwise, they would be the continuation of the preceding haplotypes.
- A homozygous genotype is always "appended" to the preceding haplotype. For example, any 0/0 or 1/1 record is always converted into 0|0 and 1|1.
- RBP attempts to phase a heterozygous genotype relative the preceding **HETEROZYGOUS** genotype for that sample. If there is sufficient read information to deduce the two haplotypes (for that sample), then the current genotype is declared phased ("/" changed to "|") and assigned a PQ that is proportional to the estimated Phred-scaled error rate. All homozygous genotypes for that sample that lie in between the two heterozygous genotypes are also assigned the same PQ value (and remain phased).
- If RBP cannot phase the heterozygous genotype, then the genotype remains with a "/", and no PQ score is assigned. This site essentially starts a new section of haplotype for this sample.

For example, consider the following records from the VCF file:

```
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  SAMP1   SAMP2
chr1    1   .   A   G   99  PASS    .   GT:GL:GQ    0/1:-100,0,-100:99  0/1:-100,0,-100:99
chr1    2   .   A   G   99  PASS    .   GT:GL:GQ:PQ 1|1:-100,0,-100:99:60
0|1:-100,0,-100:99:50
chr1    3   .   A   G   99  PASS    .   GT:GL:GQ:PQ 0|1:-100,0,-100:99:60
0|0:-100,0,-100:99:60
chr1    4   .   A   G   99  FAIL    .   GT:GL:GQ    0/1:-100,0,-100:99  0/1:-100,0,-100:99
```

```
chr1    5   .   A   G   99  PASS    .   GT:GL:GQ:PQ 0|1:-100,0,-100:99:70
1|0:-100,0,-100:99:60
chr1    6   .   A   G   99  PASS    .   GT:GL:GQ:PQ 0/1:-100,0,-100:99
1|1:-100,0,-100:99:70
chr1    7   .   A   G   99  PASS    .   GT:GL:GQ:PQ 0|1:-100,0,-100:99:80
0|1:-100,0,-100:99:70
chr1    8   .   A   G   99  PASS    .   GT:GL:GQ:PQ 0|1:-100,0,-100:99:90
0|1:-100,0,-100:99:80
```

The proper interpretation of these records is that SAMP1 has the following haplotypes at positions 1-5 of chromosome 1:

- AGAAA
- GGGAG

And two haplotypes at positions 6-8:

- AAA
- GGG

And, SAMP2 has the two haplotypes at positions 1-8:

- AAAAGGAA
- GGAAAGGG
    - Note that we have excluded the non-PASS SNP call (at chr1:4), thus assuming that both samples are homozygous reference at that site.

ReduceReads format specifications #2058

Posted on 2013-01-09 22:18:54

What is a synthetic read?

When running reduce reads, the algorithm will find regions of low variation in the genome and compress them together. To represent this compressed region, we use a synthetic read that carries all the information necessary to downstream tools to perform likelihood calculations over the reduced data.

They are called Synthetic because they are not read by a sequencer, these reads are automatically generated by the GATK and can be extremely long. In a synthetic read, each base will represent the consensus base for that genomic location. Each base will have it's consensus quality score represented in the equivalent offset in the quality score string.

Consensus Bases

ReduceReads has several filtering parameters for consensus regions. Consensus is created based on base

qualities, mapping qualities and other adjustable parameters from the command line. All filters are described in the technical documentation of reduce reads.

Consensus Quality Scores

The consensus quality score of a consensus base is essentially the mean of all bases that passed all the filters and represent an observation of that base. It is represented in the quality score field of the SAM format.

$$\frac{\sum q_i}{n}$$

n is the number of bases that contributed to the consensus base and q_i is the corresponding quality score of each base.

Insertion quality scores and Deletion quality scores (generated by BQSR) will undergo the same process and will be represented the same way.

Mapping Quality

The mapping quality of a synthetic read is a value representative of the mapping qualities of all the reads that contributed to it. This is an average of the root mean square of the mapping quality of all reads that contributed to the bases of the synthetic read. It is represented in the mapping quality score field of the SAM format.

BAD IMAGE FILE (JPEG)

Try http://cdn.vanillaforums.com/gatk.vanillaforums.com/FileUpload/16/7db81c90bec293f5be83952b74193c.jpg

where n is the number of reads and x_i is the mapping quality of each read.

Original Alignments

A synthetic read may come with up to two extra tags representing its original alignment information. Due to many filters in ReduceReads, reads are hard-clipped to the are of interest. These hard-clips are always represented in the cigar string with the H element and the length of the clipping **in genomic coordinates**. Sometimes hard clipping will make it impossible to retrieve what was the original alignment start / end of a read. In those cases,

the read will contain extra tags with integer values representing their original alignment start or end.

Here are the two integer tags:

- OP -- original alignment start
- OE -- original alignment end

For all other reads, where this can still be obtained through the cigar string (i.e. using getAlignmentStart() or getUnclippedStart()), these tags are not created.

The RR Tag

the RR tag is a tag that holds the observed depth (after filters) of every base that contributed to a reduce read. That means all bases that passed the mapping and base quality filters, and had the same observation as the one in the reduced read.

The RR tag carries an array of bytes and for increased compression, it works like this: the first number represents the depth of the first base in the reduced read. all subsequent numbers will represent the offset depth from the first base. Therefore, to calculate the depth of base "i" using the RR array, one must use :

```
RR[0] + RR[i]
```

but make sure i > 0. Here is the code we use to return the depth of the i'th base:

```
return (i==0) ? firstCount : (byte) Math.min(firstCount + offsetCount,
Byte.MAX_VALUE);
```

Using Synthetic Reads with GATK tools

The GATK is 100% compatible with synthetic reads. You can use Reduced BAM files in combination with non-reduced BAM files in any GATK analysis tools and it will work seamlessly.

Programming in the GATK

If you are programming using the GATK framework, the GATKSAMRecord class carries all the necessary functionality to use synthetic reads transparently with methods like:

- public final byte getReducedCount(final int i)
- public int getOriginalAlignmentStart()
- public int getOriginalAlignmentEnd()
- public boolean isReducedRead()

Script for sorting an input file based on a reference (SortByRef.pl) #1328

Last updated on 2012-10-18 15:31:13

This script can be used for sorting an input file based on a reference.

```
#!/usr/bin/perl -w

use strict;
use Getopt::Long;

sub usage {

    print "\nUsage:\n";
    print "sortByRef.pl [--k POS] INPUT REF_DICT\n\n";

    print " Sorts lines of the input file INFILE according\n";
    print " to the reference contig order specified by the\n";
    print " reference dictionary REF_DICT (.fai file).\n";
    print " The sort is stable. If -k option is not specified,\n";
    print " it is assumed that the contig name is the first\n";
    print " field in each line.\n\n";
    print "  INPUT      input file to sort. If '-' is specified, \n";
    print "             then reads from STDIN.\n";
    print "  REF_DICT   .fai file, or ANY file that has contigs, in the\n";
    print "             desired soting order, as its first column.\n";
    print "  --k POS :  contig name is in the field POS (1-based)\n";
    print "             of input lines.\n\n";

    exit(1);
}

my $pos = 1;
GetOptions( "k:i" => \$pos );

$pos--;

usage() if ( scalar(@ARGV) == 0 );

if ( scalar(@ARGV) != 2 ) {
    print "Wrong number of arguments\n";
    usage();
}

my $input_file = $ARGV[0];
my $dict_file = $ARGV[1];
```

```
open(DICT, "< $dict_file") or die("Can not open $dict_file: $!");

my %ref_order;

my $n = 0;
while ( <DICT> ) {
    chomp;
    my ($contig, $rest) = split "\t";
    die("Dictionary file is probably corrupt: multiple instances of contig $contig") if (
defined $ref_order{$contig} );

    $ref_order{$contig} = $n;
    $n++;
}

close DICT;
#we have loaded contig ordering now

my $INPUT;
if ($input_file eq "-" ) {
    $INPUT = "STDIN";
} else {
    open($INPUT, "< $input_file") or die("Can not open $input_file: $!");
}

my %temp_outputs;

while ( <$INPUT> ) {

    my @fields = split '\s';
    die("Specified field position exceeds the number of fields:\n$_")
        if ( $pos >= scalar(@fields) );

    my $contig = $fields[$pos];
    if ( $contig =~ m/:/ ) {
        my @loc = split(/:/, $contig);
        # print $contig . " " . $loc[0] . "\n";
        $contig = $loc[0]
    }
    chomp $contig if ( $pos == scalar(@fields) - 1 ); # if last field in line

    my $order;
    if ( defined $ref_order{$contig} ) { $order = $ref_order{$contig}; }
    else {
        $order = $n; # input line has contig that was not in the dict;
        $n++; # this contig will go at the end of the output,
```

```
            # after all known contigs
    }

    my $fhandle;
    if ( defined $temp_outputs{$order} ) { $fhandle = $temp_outputs{$order} }
    else {
        #print "opening $order $$ $_\n";
        open( $fhandle, " > /tmp/sortByRef.$$.$order.tmp" ) or
            die ( "Can not open temporary file $order: $!");
        $temp_outputs{$order} = $fhandle;
    }

    # we got the handle to the temp file that keeps all
    # lines with contig $contig

    print $fhandle $_; # send current line to its corresponding temp file
}

close $INPUT;

foreach my $f ( values %temp_outputs ) { close $f; }

# now collect back into single output stream:

for ( my $i = 0 ; $i < $n ; $i++ ) {
    # if we did not have any lines on contig $i, then there's
    # no temp file and nothing to do
    next if ( ! defined $temp_outputs{$i} ) ;

    my $f;
    open ( $f, "< /tmp/sortByRef.$$.$i.tmp" );
    while ( <$f> ) { print ; }
    close $f;

    unlink "/tmp/sortByRef.$$.$i.tmp";


}
```

Using CombineVariants #53

Last updated on 2013-01-12 22:06:29

1. About CombineVariants

This tool combines VCF records from different sources. Any (unique) name can be used to bind your rod data and any number of sources can be input. This tool currently supports two different combination types for each of variants (the first 8 fields of the VCF) and genotypes (the rest)

For a complete, detailed argument reference, refer to the GATK document page here.

2. Logic for merging records across VCFs

CombineVariants will include a record at every site in all of your input VCF files, and annotate which input ROD bindings the record is present, pass, or filtered in in the set attribute in the `INFO` field (see below). In effect, CombineVariants always produces a union of the input VCFs. However, any part of the Venn of the N merged VCFs can be exacted using JEXL expressions on the set attribute using SelectVariants. If you want to extract just the records in common between two VCFs, you would first CombineVariants the two files into a single VCF, and then run SelectVariants to extract the common records with `-select 'set == "Intersection"'`, as worked out in the detailed example below.

3. Handling PASS/FAIL records at the same site in multiple input files

The `-filteredRecordsMergeType` argument determines how CombineVariants handles sites where a record is present in multiple VCFs, but it is filtered in some and unfiltered in others, as described in the Tech Doc page for the tool.

4. Understanding the set attribute

The set `INFO` field indicates which call set the variant was found in. It can take on a variety of values indicating the exact nature of the overlap between the call sets. Note that the values are generalized for multi-way combinations, but here we describe only the values for 2 call sets being combined.

- `set=Intersection` : occurred in both call sets, not filtered out
- `set=NAME` : occurred in the call set `NAME` only
- `set=NAME1-filteredInNAME` : occurred in both call sets, but was not filtered in `NAME1` but was filtered in `NAME2`
- `set=filteredInAll` : occurred in both call sets, but was filtered out of both

For three or more call sets combinations, you can see records like `NAME1-NAME2` indicating a variant occurred in both `NAME1` and `NAME2` but not all sets.

5. Changing the set key

You can use `-setKey foo` to change the `set=XXX` tag to `foo=XXX` in your output. Additionally, `-setKey null` stops the set tag=value pair from being emitted at all.

6. Minimal VCF output

Add the `-minimalVCF` argument to CombineVariants if you want to eliminate unnecessary information from the `INFO` field and genotypes. The only fields emitted will be `GT:GQ` for genotypes and the `keySet` for `INFO`

An even more extreme output format is `-sites_only`, a general engine capability, where the genotypes for all samples are completely stripped away from the output format. Enabling this option results in a significant performance speedup as well.

7. Combining Variant Calls with a minimum set of input sites

Add the `-minN` (or `--minimumN`) command, followed by an integer if you want to only output records present in at least N input files. Useful, for example in combining several data sets where we only want to keep sites present in for example at least 2 of them (in which case `-minN 2` should be added to the command line).

8. Example: intersecting two VCFs

In the following example, we use CombineVariants and SelectVariants to obtain only the sites in common between the OMNI 2.5M and HapMap3 sites in the GSA bundle.

```
java -Xmx2g -jar dist/GenomeAnalysisTK.jar -T CombineVariants -R
bundle/b37/human_g1k_v37.fasta -L 1:1-1,000,000 -V:omni
bundle/b37/1000G_omni2.5.b37.sites.vcf -V:hm3 bundle/b37/hapmap_3.3.b37.sites.vcf -o
union.vcf
java -Xmx2g -jar dist/GenomeAnalysisTK.jar -T SelectVariants -R
~/Desktop/broadLocal/localData/human_g1k_v37.fasta -L 1:1-1,000,000 -V:variant union.vcf
-select 'set == ";Intersection";' -o intersect.vcf
```

This results in two vcf files, which look like:

```
==> union.vcf <==
1       990839  SNP1-980702     C       T       .       PASS
AC=150;AF=0.05384;AN=2786;CR=100.0;GentrainScore=0.7267;HW=0.0027632264;set=Intersection
1       990882  SNP1-980745     C       T       .       PASS
CR=99.79873;GentrainScore=0.7403;HW=0.005225421;set=omni
1       990984  SNP1-980847     G       A       .       PASS
CR=99.76005;GentrainScore=0.8406;HW=0.26163524;set=omni
1       992265  SNP1-982128     C       T       .       PASS
CR=100.0;GentrainScore=0.7412;HW=0.0025895447;set=omni
1       992819  SNP1-982682     G       A       .       id50
CR=99.72961;GentrainScore=0.8505;HW=4.811053E-17;set=FilteredInAll
1       993987  SNP1-983850     T       C       .       PASS
CR=99.85935;GentrainScore=0.8336;HW=9.959717E-28;set=omni
1       994391  rs2488991       G       T       .       PASS
AC=1936;AF=0.69341;AN=2792;CR=99.89378;GentrainScore=0.7330;HW=1.1741E-41;set=filterInomni-h
m3
1       996184  SNP1-986047     G       A       .       PASS
CR=99.932205;GentrainScore=0.8216;HW=3.8830226E-6;set=omni
1       998395  rs7526076       A       G       .       PASS
AC=2234;AF=0.80187;AN=2786;CR=100.0;GentrainScore=0.8758;HW=0.67373306;set=Intersection
1       999649  SNP1-989512     G       A       .       PASS
CR=99.93262;GentrainScore=0.7965;HW=4.9767335E-4;set=omni
```

```
==> intersect.vcf <==
1       950243  SNP1-940106     A       C       .       PASS
AC=826;AF=0.29993;AN=2754;CR=97.341675;GentrainScore=0.7311;HW=0.15148845;set=Intersection
1       957640  rs6657048       C       T       .       PASS
AC=127;AF=0.04552;AN=2790;CR=99.86667;GentrainScore=0.6806;HW=2.286109E-4;set=Intersection
1       959842  rs2710888       C       T       .       PASS
AC=654;AF=0.23559;AN=2776;CR=99.849;GentrainScore=0.8072;HW=0.17526293;set=Intersection
1       977780  rs2710875       C       T       .       PASS
AC=1989;AF=0.71341;AN=2788;CR=99.89077;GentrainScore=0.7875;HW=2.9912625E-32;set=Intersectio
n
1       985900  SNP1-975763     C       T       .       PASS
AC=182;AF=0.06528;AN=2788;CR=99.79926;GentrainScore=0.8374;HW=0.017794203;set=Intersection
1       987200  SNP1-977063     C       T       .       PASS
AC=1956;AF=0.70007;AN=2794;CR=99.45917;GentrainScore=0.7914;HW=1.413E-42;set=Intersection
1       987670  SNP1-977533     T       G       .       PASS
AC=2485;AF=0.89196;AN=2786;CR=99.51427;GentrainScore=0.7005;HW=0.24214932;set=Intersection
1       990417  rs2465136       T       C       .       PASS
AC=1113;AF=0.40007;AN=2782;CR=99.7599;GentrainScore=0.8750;HW=8.595538E-5;set=Intersection
1       990839  SNP1-980702     C       T       .       PASS
AC=150;AF=0.05384;AN=2786;CR=100.0;GentrainScore=0.7267;HW=0.0027632264;set=Intersection
1       998395  rs7526076       A       G       .       PASS
AC=2234;AF=0.80187;AN=2786;CR=100.0;GentrainScore=0.8758;HW=0.67373306;set=Intersection
```

Using DepthOfCoverage to find out how much sequence data you have #40

Last updated on 2013-08-23 18:36:27

See this announcement regarding our plans for support of DepthOfCoverage and DiagnoseTargets. If you find that there are functionalities missing in either tool, leave us a comment and we will consider adding them.

For a complete, detailed argument reference, refer to the GATK document page here.

Introduction

DepthOfCoverage is a coverage profiler for a (possibly multi-sample) bam file. It uses a granular histogram that can be user-specified to present useful aggregate coverage data. It reports the following metrics over the entire .bam file:

- Total, mean, median, and quartiles for each partition type: aggregate
- Total, mean, median, and quartiles for each partition type: for each interval
- A series of histograms of the number of bases covered to Y depth for each partition type (granular; e.g. Y can be a range, like 16 to 22)
- A matrix of counts of the number of intervals for which at least Y samples and/or read groups had a median coverage of at least X

- A matrix of counts of the number of bases that were covered to at least X depth, in at least Y groups (e.g. # of loci with â‰¥15x coverage for â‰¥12 samples)

- A matrix of proportions of the number of bases that were covered to at least X depth, in at least Y groups (e.g. proportion of loci with â‰¥18x coverage for â‰¥15 libraries)

Because the common question "What proportion of my targeted bases are well-powered to discover SNPs?" is answered by the last matrix on the above list, it is strongly recommended that this walker be run on all samples simultaneously.

For humans, DepthOfCoverage can also be configured to output these statistics aggregated over genes, by providing it with a RefSeq ROD.

DepthOfCoverage also outputs, by default, the total coverage at every locus, and the coverage per sample and/or read group. This behavior can optionally be turned off, or switched to base count mode, where base counts will be output at each locus, rather than total depth.

Coverage by Gene

To get a summary of coverage by each gene, you may supply a refseq (or alternative) gene list via the argument

```
-geneList /path/to/gene/list.txt
```

The provided gene list must be of the following format:

```
585     NM_001005484    chr1    +       58953   59871   58953   59871   1       58953,
59871,  0       OR4F5   cmpl    cmpl    0,
587     NM_001005224    chr1    +       357521  358460  357521  358460  1       357521,
358460, 0       OR4F3   cmpl    cmpl    0,
587     NM_001005277    chr1    +       357521  358460  357521  358460  1       357521,
358460, 0       OR4F16  cmpl    cmpl    0,
587     NM_001005221    chr1    +       357521  358460  357521  358460  1       357521,
358460, 0       OR4F29  cmpl    cmpl    0,
589     NM_001005224    chr1    -       610958  611897  610958  611897  1       610958,
611897, 0       OR4F3   cmpl    cmpl    0,
589     NM_001005277    chr1    -       610958  611897  610958  611897  1       610958,
611897, 0       OR4F16  cmpl    cmpl    0,
589     NM_001005221    chr1    -       610958  611897  610958  611897  1       610958,
611897, 0       OR4F29  cmpl    cmpl    0,
```

If you are on the broad network, the properly-formatted file containing refseq genes and transcripts is located at

```
/humgen/gsa-hpprojects/GATK/data/refGene.sorted.txt
```

If you supply the -geneList argument, DepthOfCoverage will output an additional summary file that looks as follows:

```
Gene_Name     Total_Cvg       Avg_Cvg       Sample_1_Total_Cvg    Sample_1_Avg_Cvg
```

```
Sample_1_Cvg_Q3       Sample_1_Cvg_Median      Sample_1_Cvg_Q1
SORT1    594710  238.27  594710  238.27  165     245     330
NOTCH2  3011542 357.84  3011542 357.84  222     399     &gt;500
LMNA    563183  186.73  563183  186.73  116     187     262
NOS1AP  513031  203.50  513031  203.50  91      191     290
```

Note that the gene coverage will be aggregated only over samples (not read groups, libraries, or other types). The `-geneList` argument also requires specific intervals within genes to be given (say, the particular exons you are interested in, or the entire gene), and it functions by aggregating coverage from the interval level to the gene level, by referencing each interval to the gene in which it falls. Because by-gene aggregation looks for intervals that overlap genes, `-geneList` is ignored if `-omitIntervals` is thrown.

Using RefSeq data #1329

Last updated on 2012-09-28 16:40:31

1. About the RefSeq Format

From the NCBI RefSeq website

```
The Reference Sequence (RefSeq) collection aims to provide a comprehensive, integrated,
non-redundant, well-annotated set of sequences, including genomic DNA, transcripts, and
proteins. RefSeq is a foundation for medical, functional, and diversity studies; they
provide a stable reference for genome annotation, gene identification and characterization,
mutation and polymorphism analysis (especially RefSeqGene records), expression studies, and
comparative analyses.
```

2. In the GATK

The GATK uses RefSeq in a variety of walkers, from indel calling to variant annotations. There are many file format flavors of ReqSeq; we've chosen to use the table dump available from the UCSC genome table browser.

3. Generating RefSeq files

Go to the UCSC genome table browser. There are many output options, here are the changes that you'll need to make:

```
clade:    Mammal
genome:   Human
assembly: ''choose the appropriate assembly for the reference you're using''
group:    Genes abd Gene Prediction Tracks
track:    RefSeq Genes
table:    refGene
region:   ''choose the genome option''
```

Choose a good output filename, something like `geneTrack.refSeq`, and click the `get output` button. You now have your initial RefSeq file, which will not be sorted, and will contain non-standard contigs. To run with the

GATK, contigs other than the standard 1-22,X,Y,MT must be removed, and the file sorted in karyotypic order. This can be done with a combination of grep, sort, and a script called `sortByRef.pl` that is available here.

4. Running with the GATK

You can provide your RefSeq file to the GATK like you would for any other ROD command line argument. The line would look like the following:

```
-[arg]:REFSEQ /path/to/refSeq
```

Using the filename from above.

Warning:

The GATK automatically adjusts the start and stop position of the records from zero-based half-open intervals (UCSC standard) to one-based closed intervals.

For example:

```
The first 19 bases in Chromsome one:
Chr1:0-19 (UCSC system)
Chr1:1-19 (GATK)
```

All of the GATK output is also in this format, so if you're using other tools or scripts to process RefSeq or GATK output files, you should be aware of this difference.

Using SelectVariants #54

Last updated on 2013-04-04 00:53:42

Introduction

SelectVariants is a GATK tool used to subset a VCF file by many arbitrary criteria listed in the command line options below. The output VCF wiil have the AN (number of alleles), AC (allele count), AF (allele frequency), and DP (depth of coverage) annotations updated as necessary to accurately reflect the file's new contents.

Select Variants operates on VCF files (ROD Tracks) provided in the command line using the GATK's built in `--variant` option. You can provide multiple tracks for Select Variants but at least one must be named 'variant' and this will be the file all your analysis will be based of. Other tracks can be named as you please. Options requiring a reference to a ROD track name will use the track name provided in the -B option to refer to the correct VCF file (e.g. --discordance / --concordance). All other analysis will be done in the 'variant' track.

Often, a VCF containing many samples and/or variants will need to be subset in order to facilitate certain analyses (e.g. comparing and contrasting cases vs. controls; extracting variant or non-variant loci that meet certain requirements, displaying just a few samples in a browser like IGV, etc.). SelectVariants can be used for this purpose. Given a single VCF file, one or more samples can be extracted from the file (based on a complete sample name or a pattern match). Variants can be further selected by specifying criteria for inclusion, i.e. "DP >

1000" (depth of coverage greater than 1000x), "AF < 0.25" (sites with allele frequency less than 0.25). These JEXL expressions are documented here in the FAQ article on JEXL expressions; it is particularly important to note the section on working with complex expressions.

Command-line arguments

For a complete, detailed argument reference, refer to the GATK document page here.

How do the AC, AF, AN, and DP fields change?

Let's say you have a file with three samples. The numbers before the ":" will be the genotype (0/0 is hom-ref, 0/1 is het, and 1/1 is hom-var), and the number after will be the depth of coverage.

```
BOB         MARY         LINDA
1/0:20      0/0:30       1/1:50
```

In this case, the INFO field will say AN=6, AC=3, AF=0.5, and DP=100 (in practice, I think these numbers won't necessarily add up perfectly because of some read filters we apply when calling, but it's approximately right).

Now imagine I only want a file with the samples "BOB" and "MARY". The new file would look like:

```
BOB         MARY
1/0:20      0/0:30
```

The INFO field will now have to change to reflect the state of the new data. It will be AN=4, AC=1, AF=0.25, DP=50.

Let's pretend that MARY's genotype wasn't 0/0, but was instead "./." (no genotype could be ascertained). This would look like

```
BOB         MARY
1/0:20      ./.:.
```

with AN=2, AC=1, AF=0.5, and DP=20.

Subsetting by sample and ALT alleles

SelectVariants now keeps (r5832) the alt allele, even if a record is AC=0 after subsetting the site down to selected samples. For example, when selecting down to just sample NA12878 from the OMNI VCF in 1000G (1525 samples), the resulting VCF will look like:

```
1       82154   rs4477212       A       G       .       PASS
AC=0;AF=0.00;AN=2;CR=100.0;DP=0;GentrainScore=0.7826;HW=1.0     GT:GC   0/0:0.7205
1       534247  SNP1-524110     C       T       .       PASS
AC=0;AF=0.00;AN=2;CR=99.93414;DP=0;GentrainScore=0.7423;HW=1.0  GT:GC   0/0:0.6491
1       565286  SNP1-555149     C       T       .       PASS
AC=2;AF=1.00;AN=2;CR=98.8266;DP=0;GentrainScore=0.7029;HW=1.0   GT:GC   1/1:0.3471
1       569624  SNP1-559487     T       C       .       PASS
```

```
AC=2;AF=1.00;AN=2;CR=97.8022;DP=0;GentrainScore=0.8070;HW=1.0   GT:GC   1/1:0.3942
```

Although NA12878 is 0/0 at the first sites, ALT allele is preserved in the VCF record. This is the correct behavior, as reducing samples down shouldn't change the character of the site, only the AC in the subpopulation. This is related to the tricky issue of isPolymorphic() vs. isVariant().

- isVariant => is there an ALT allele?
- isPolymorphic => is some sample non-ref in the samples?

In part this is complicated as the semantics of sites-only VCFs, where ALT = . is used to mean not-polymorphic. Unfortunately, I just don't think there's a consistent convention right now, but it might be worth at some point to adopt a single approach to handling this.

For clarity, in previous versions of SelectVariants, the first two monomorphic sites lose the ALT allele, because NA12878 is hom-ref at this site, resulting in VCF that looks like:

```
1       82154   rs4477212       A       .       .       PASS
AC=0;AF=0.00;AN=2;CR=100.0;DP=0;GentrainScore=0.7826;HW=1.0     GT:GC   0/0:0.7205
1       534247  SNP1-524110     C       .       .       PASS
AC=0;AF=0.00;AN=2;CR=99.93414;DP=0;GentrainScore=0.7423;HW=1.0  GT:GC   0/0:0.6491
1       565286  SNP1-555149     C       T       .       PASS
AC=2;AF=1.00;AN=2;CR=98.8266;DP=0;GentrainScore=0.7029;HW=1.0   GT:GC   1/1:0.3471
1       569624  SNP1-559487     T       C       .       PASS
AC=2;AF=1.00;AN=2;CR=97.8022;DP=0;GentrainScore=0.8070;HW=1.0   GT:GC   1/1:0.3942
```

If you really want a VCF without monomorphic sites, use the option to drop monomorphic sites after subsetting.

Known issues

Some VCFs may have repeated header entries with the same key name, for instance:

```
##fileformat=VCFv3.3
##FILTER=ABFilter,&quot;AB &gt; 0.75&quot;
##FILTER=HRunFilter,&quot;HRun &gt; 3.0&quot;
##FILTER=QDFilter,&quot;QD &lt; 5.0&quot;
##UG_bam_file_used=file1.bam
##UG_bam_file_used=file2.bam
##UG_bam_file_used=file3.bam
##UG_bam_file_used=file4.bam
##UG_bam_file_used=file5.bam
##source=UnifiedGenotyper
##source=VariantFiltration
##source=AnnotateVCFwithMAF
...
```

Here, the "UG_bam_file_used" and "source" header lines appear multiple times. When SelectVariants is run on

such a file, the program will emit warnings that these repeated header lines are being discarded, resulting in only the first instance of such a line being written to the resulting VCF. This behavior is not ideal, but expected under the current architecture.

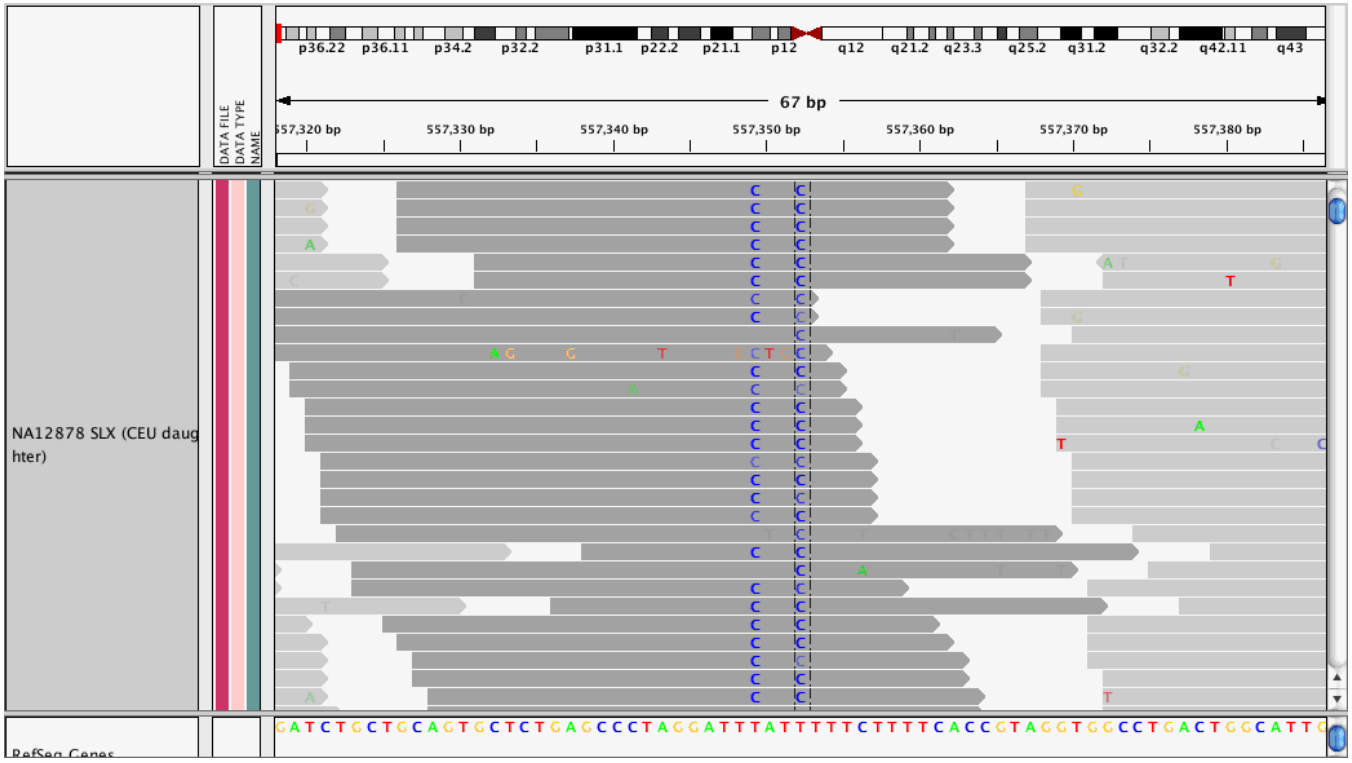
Additional information

For information on how to construct regular expressions for use with this tool, see the "Summary of regular-expression constructs" section here.

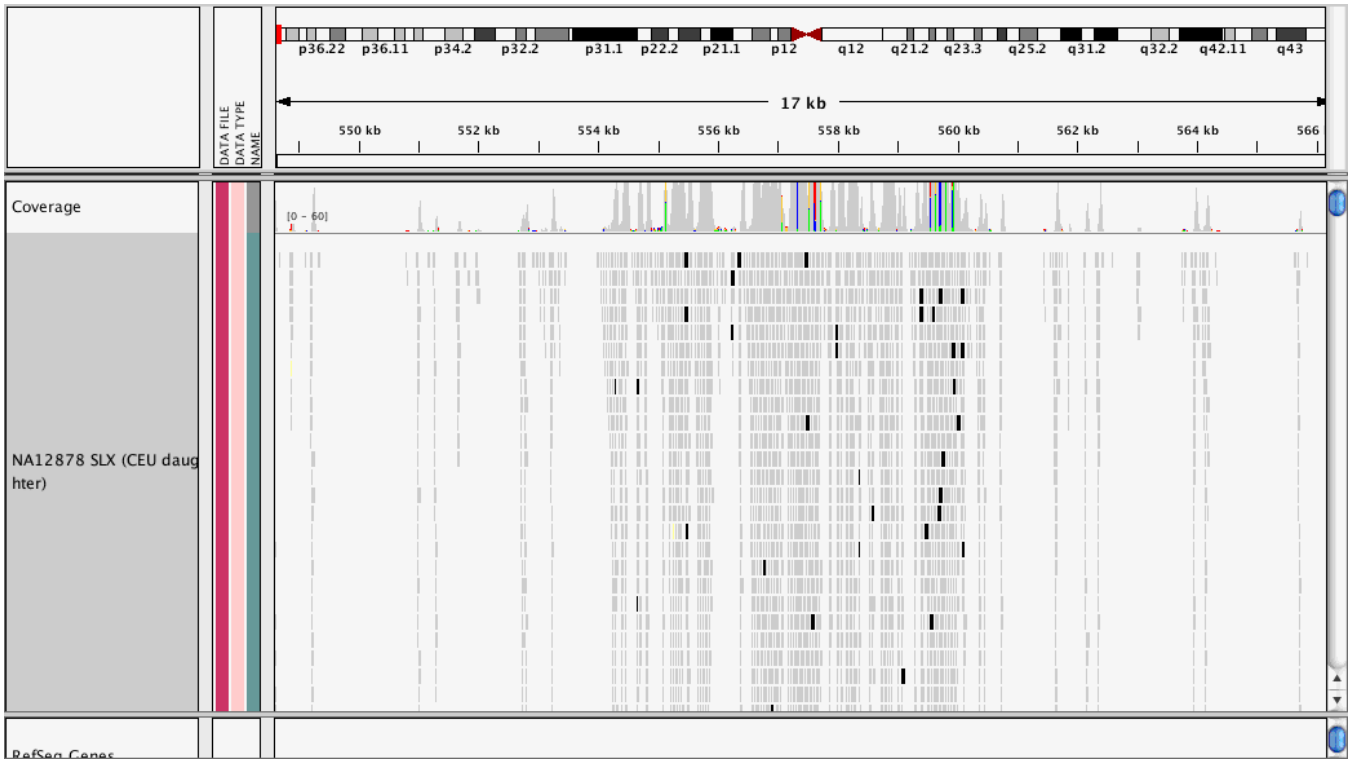
Using Variant Annotator #49

Last updated on 2012-12-11 20:45:20

2 SNPs with significant strand bias



Several SNPs with excessive coverage

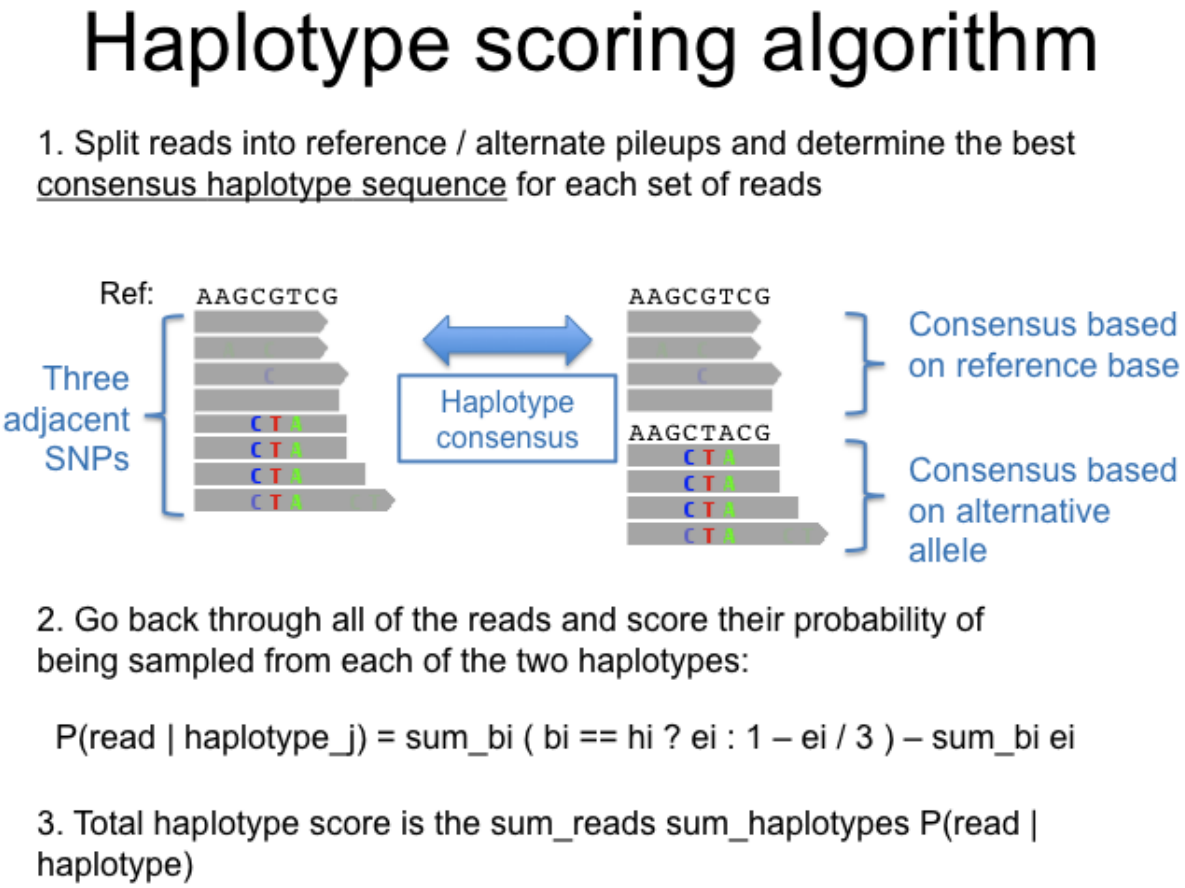


For a complete, detailed argument reference, refer to the GATK document page here.

Introduction

In addition to true variation, variant callers emit a number of false-positives. Some of these false-positives can be detected and rejected by various statistical tests. VariantAnnotator provides a way of annotating variant calls as preparation for executing these tests.

Description of the haplotype score annotation



Examples of Available Annotations

The list below is not comprehensive. Please use the `--list` argument to get a list of all possible annotations available. Also, see the FAQ article on understanding the Unified Genotyper's VCF files for a description of some of the more standard annotations.

- BaseQualityRankSumTest (BaseQRankSum)
- DepthOfCoverage (DP)
- FisherStrand (FS)
- HaplotypeScore (HaplotypeScore)
- MappingQualityRankSumTest (MQRankSum)
- MappingQualityZero (MQ0)
- QualByDepth (QD)
- ReadPositionRankSumTest (ReadPosRankSum)
- RMSMappingQuality (MQ)
- SnpEff: Add genomic annotations using the third-party tool SnpEff with VariantAnnotator

Note that technically the VariantAnnotator does not require reads (from a BAM file) to run; if no reads are provided, only those Annotations which don't use reads (e.g. Chromosome Counts) will be added. But most Annotations do require reads. **When running the tool we recommend that you add the** `-L` argument with the variant rod to your command line for efficiency and speed.

Using Variant Filtration #51

Last updated on 2012-11-29 19:46:33

VariantFiltration

For a complete, detailed argument reference, refer to the GATK document page here.

The documentation for Using JEXL expressions within the GATK contains very important information about limitations of the filtering that can be done; in particular please note the section on working with complex expressions.

Filtering Individual Genotypes

One can now filter individual samples/genotypes in a VCF based on information from the `FORMAT` field: Variant Filtration will add the sample-level `FT` tag to the `FORMAT` field of filtered samples (this does not affect the record's `FILTER` tag). This is still a work in progress and isn't quite as flexible and powerful yet as we'd like it to be. For now, one can filter based on most fields as normal (e.g. `GQ < 5.0`), but the `GT` (genotype) field is an exception. We have put in convenience methods so that one can now filter out hets (`isHet == 1`), refs (`isHomRef == 1`), or homs (`isHomVar == 1`).

Using VariantEval #48

Last updated on 2013-03-19 17:57:23

For a complete, detailed argument reference, refer to the technical documentation page.

Modules

You can find detailed information about the various modules here.

Stratification modules

- AlleleFrequency
- AlleleCount
- CompRod
- Contig
- CpG

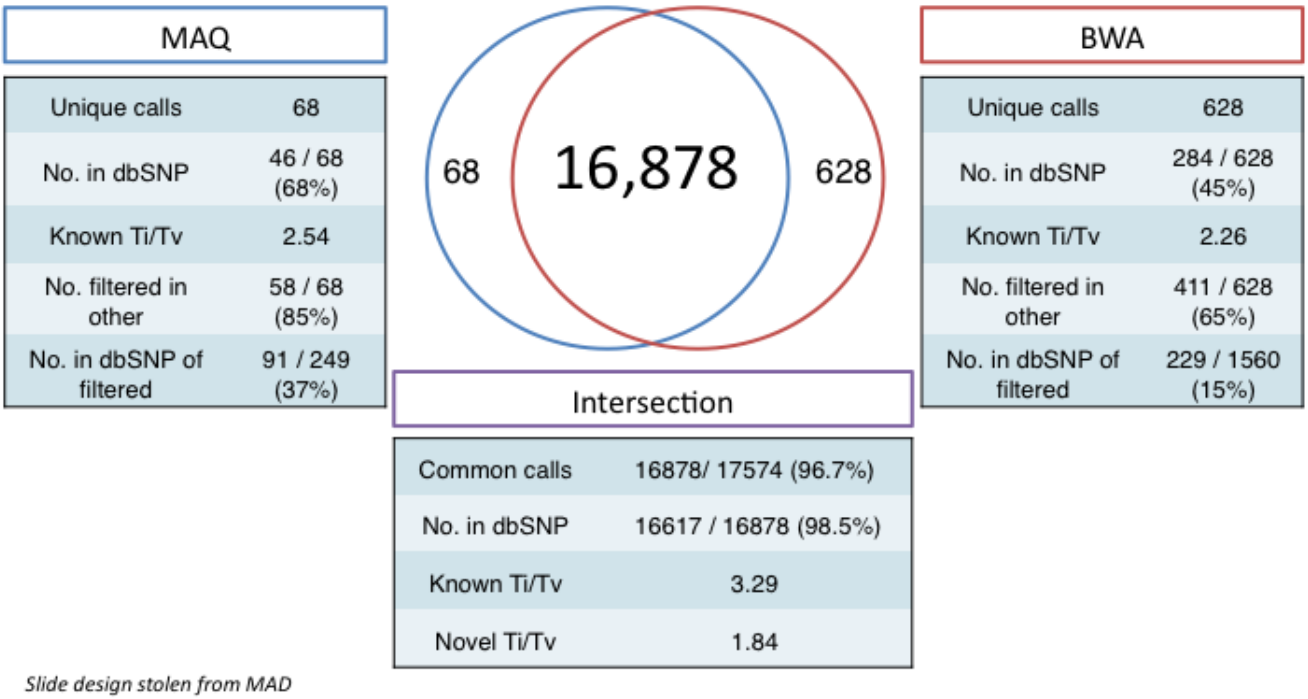
- Degeneracy
- EvalRod
- Filter
- FunctionalClass
- JexlExpression
- Novelty
- Sample

Evaluation modules

- CompOverlap
- CountVariants

Note that the GenotypeConcordance module has been rewritten as a separate walker tool (see its Technical Documentation page).

A useful analysis using VariantEval



We in GSA often find ourselves performing an analysis of 2 different call sets. For SNPs, we often show the overlap of the sets (their "venn") and the relative dbSNP rates and/or transition-transversion ratios. The picture provided is an example of such a slide and is easy to create using VariantEval. Assuming you have 2 filtered VCF callsets named 'foo.vcf' and 'bar.vcf', there are 2 quick steps.

Combine the VCFs

```
java -jar GenomeAnalysisTK.jar \
     -R ref.fasta \
     -T CombineVariants \
     -V:FOO foo.vcf \
     -V:BAR bar.vcf \
     -priority FOO,BAR \
     -o merged.vcf
```

Run VariantEval

```
java -jar GenomeAnalysisTK.jar \
     -T VariantEval \
     -R ref.fasta \
```

```
-D dbsnp.vcf \
-select 'set=="Intersection"' -selectName Intersection \
-select 'set=="FOO"' -selectName FOO \
-select 'set=="FOO-filterInBAR"' -selectName InFOO-FilteredInBAR \
-select 'set=="BAR"' -selectName BAR \
-select 'set=="filterInFOO-BAR"' -selectName InBAR-FilteredInFOO \
-select 'set=="FilteredInAll"' -selectName FilteredInAll \
-o merged.eval.gatkreport \
-eval merged.vcf \
-l INFO
```

Checking the possible values of 'set'

It is wise to check the actual values for the set names present in your file before writing complex VariantEval commands. An easy way to do this is to extract the value of the set fields and then reduce that to the unique entries, like so:

```
java -jar GenomeAnalysisTK.jar -T VariantsToTable -R ref.fasta -V merged.vcf -F set -o
fields.txt
grep -v 'set' fields.txt | sort | uniq -c
```

This will provide you with a list of all of the possible values for 'set' in your VCF so that you can be sure to supply the correct select statements to VariantEval.

Reading the VariantEval output file

The VariantEval output is formatted as a GATKReport.

Understanding Genotype Concordance values from Variant Eval

The VariantEval genotype concordance module emits information the relationship between the eval calls and genotypes and the comp calls and genotypes. The following three slides provide some insight into three key metrics to assess call sensitivity and concordance between genotypes.

```
##:GATKReport.v0.1 GenotypeConcordance.sampleSummaryStats : the concordance statistics
summary for each sample
GenotypeConcordance.sampleSummaryStats  CompRod   CpG      EvalRod  JexlExpression  Novelty
percent_comp_ref_called_var  percent_comp_het_called_het  percent_comp_het_called_var
percent_comp_hom_called_hom  percent_comp_hom_called_var  percent_non-reference_sensitivity
percent_overall_genotype_concordance  percent_non-reference_discrepancy_rate
GenotypeConcordance.sampleSummaryStats  compOMNI  all      eval     none            all
0.78                          97.65                         98.39                        99.13
                      99.44                          98.80                             99.09
                               3.60
```

The key outputs:

- `percent_overall_genotype_concordance`
- `percent_non_ref_sensitivity_rate`
- `percent_non_ref_discrepancy_rate`

All defined below.

Non-reference sensitivity

Genotype of comparison calls

Genotype of evaluation calls	A/A	A/B	B/B	./.
A/A	1	2	3	4
A/B	5	6	7	8
B/B	9	10	11	12
./.	13	14	15	16

A is ref allele, B is alt allele, . is no call

- Blue elements / red area elements
- Measures fraction of sites called variant (A/B or B/B) in comparison that are also called variant in evaluation data
- Penalizes hom-ref and no-calls at variant sites

(6+7+10+11) /
(2+3+6+7+10+11+14+15)
= 34/ 68

Non-reference discrepancy rate

Genotype of comparison calls

Genotype of evaluation calls	A/A	A/B	B/B	./.
A/A	1	2	3	4
A/B	5	6	7	8
B/B	9	10	11	12
./.	13	14	15	16

A is ref allele, B is alt allele, . is no call

- Blue elements / red area elements
- Measures accuracy of genotype calls at sites called by both sets
  - Excludes concordant A/A genotypes since these are often large in number and easier to get correct
- Good metric for assaying accuracy of genotype calls

(2+3+5+7+9+10) /
(2+3+5+6+7+9+10+11)
= 36 / 53

Overall genotype concordance

Genotype of comparison calls

Genotype of evaluation calls	A/A	A/B	B/B	./.
A/A	1	2	3	4
A/B	5	6	7	8
B/B	9	10	11	12
./.	13	14	15	16

A is ref allele, B is alt allele, . is no call

- Blue elements / red area elements
- Measures accuracy of genotype calls at all loci
  - Excludes no-calls in either set
  - Often biased by concordant A/A loci

1 + 6 + 11 / (1+2+3+5+6+7+9+10+11)
= 18 / 54

Not recommended for routine analysis

Example

Genotype of comparison calls

Genotype of evaluation calls	A/A	A/B	B/B	./.
A/A	1	1	1	1
A/B	1	1	1	1
B/B	1	1	1	1
./.	1	1	1	1

A is ref allele, B is alt allele, . is no call

- Non-reference sensitivity:
  - 4 / 8 = 50%
- Non-reference discrepancy rate:
  - 6 / 8 = 75%
- Overall genotype concordance
  - 3 / 9 = 33%

Example VCFs in /humgen/gsa-hpprojects/GATK/data/Validation_Data/GenotypeConcordance*.vcf implement this table

Using the Somatic Indel Detector #35

Last updated on 2012-09-28 18:06:11

Note that the Somatic Indel Detector was previously called Indel Genotyper V2.0

For a complete, detailed argument reference, refer to the GATK document page here.

Calling strategy

The Somatic Indel Detector can be run in two modes: single sample and paired sample. In the former mode, exactly one input bam file should be given, and indels in that sample are called. In the paired mode, the calls are made in the tumor sample, but in addition to that the differential signal is sought between the two samples (e.g. somatic indels present in tumor cell DNA but not in the normal tissue DNA). In the paired mode, the genotyper makes an initial call in the tumor sample in the same way as it would in the single sample mode; the call,

however, is then compared to the normal sample. If any evidence (even very weak, so that it would not trigger a call in single sample mode) for the event is found in the normal, the indel is annotated as germline. Only when the minimum required coverage in the normal sample is achieved and there is no evidence in the normal sample for the event called in the tumor is the indel annotated as somatic.

The calls in both modes (recall that in paired mode the calls are made in tumor sample only and are simply annotated according to the evidence in the matching normal) are performed based on a set of simple thresholds. Namely, all distinct events (indels) at the given site are collected, along with the respective counts of alignments (reads) supporting them. The putative call is the majority vote *consensus* (i.e. the indel that has the largest count of reads supporting it). This call is accepted if 1) there is enough coverage (as well as enough coverage in matching normal sample in paired mode); 2) reads supporting the consensus indel event constitute a sufficiently large fraction of the total coverage at the site; 3) reads supporting the consensus indel event constitute a sufficiently large fraction of all the reads supporting *any* indel at the site. See details in the Arguments section of the tool documentation.

Theoretically, the Somatic Indel Detector can be run directly on the aligned short read sequencing data. However, it does not perform any deep algorithmic tasks such as searching for misplaced indels close to a given one, or correcting read misalignments given the presence of an indel in another read, etc. Instead, it assumes that all the evidence for indels (all the reads that support it), for the presence of the matching event in normal etc is already in the input and performs simple counting. It is thus highly, *HIGHLY* recommended to run the Somatic Indel Detector on "cleaned" bam files, after performing Local realignment around indels.

Output

Brief output file (specified with -bed option) will look as follows:

```
chr1    556817  556817  +G:3/7
chr1    3535035 3535054 -TTCTGGGAGCTCCTCCCCC:9/21
chr1    3778838 3778838 +A:15/48
...
```

This is a .bed track that can be loaded into UCSC browser or IGV browser, the event itself and the <count of supporting reads>/<total coverage> are reported in the 'name' field of the file. The event locations on the chromosomes are 1-based, and the convention is that all events (*both* insertions and deletions) are assigned to the base on the reference *immediately preceding the event* (second column). The third column is the stop position of the event on the reference, or strictly speaking the base immediately preceding the first base on the reference after the event: the last deleted base for deletions, or the same base as the start position for insertions. For instance, the first line in the above example specifies an insertion (+G) supported by 3 reads out of 7 (i.e. total coverage at the site is 7x) that occurs immediately after genomic position chr1:556817. The next line specifies a 19 bp deletion -TTCTGGGAGCTCCTCCCCC supported by 9 reads (total coverage 21x) occuring at (after) chr1:3535035 (the first and last *deleted* bases are 3535035+1=3535036 and 3535054, respectively).

Note that in the paired mode all calls made in tumor (both germline and somatic) will be printed into the brief output without further annotations.

The detailed (verbose) output option is kept for backward compatibility with post-processing tools that might have been developed to work with older versions of the IndelGenotyperV2. All the information described below is now also recorded into the vcf output file, so the verbose text output is completely redundant, except for genomic annotations (if --refseq is used). Generated vcf file can be annotated separately using VCF post-processing tools.

The detailed output (-verbose) will contain additional statistics characterizing the alignments around each called event, SOMATIC/GERMLINE annotations (in paired mode), as well as genomic annotations (when --refseq is used). The verbose output lines matching the three lines from the example above could look like this (note that the long lines are wrapped here, the actual output file contains one line per event):

```
chr1    556817  556817  +G      N_OBS_COUNTS[C/A/T]:0/0/52      N_AV_MM[C/R]:0.00/5.27
N_AV_MAPQ[C/R]:0.00/35.17    \
                N_NQS_MM_RATE[C/R]:0.00/0.08    N_NQS_AV_QUAL[C/R]:0.00/23.74
N_STRAND_COUNTS[C/C/R/R]:0/0/32/20  \
                T_OBS_COUNTS[C/A/T]:3/3/7       T_AV_MM[C/R]:2.33/5.50
T_AV_MAPQ[C/R]:66.00/24.75   \
                T_NQS_MM_RATE[C/R]:0.05/0.08    T_NQS_AV_QUAL[C/R]:20.26/11.61
T_STRAND_COUNTS[C/C/R/R]:3/0/2/2 \
                SOMATIC GENOMIC
chr1    3535035 3535054 -TTCTGGGAGCTCCTCCCCC  N_OBS_COUNTS[C/A/T]:3/3/6
N_AV_MM[C/R]:3.33/2.67  N_AV_MAPQ[C/R]:73.33/99.00 \
                N_NQS_MM_RATE[C/R]:0.00/0.00    N_NQS_AV_QUAL[C/R]:29.27/31.83
N_STRAND_COUNTS[C/C/R/R]:0/3/0/3  \
                T_OBS_COUNTS[C/A/T]:9/9/21      T_AV_MM[C/R]:1.56/0.17
T_AV_MAPQ[C/R]:88.00/99.00    \
                T_NQS_MM_RATE[C/R]:0.02/0.00   T_NQS_AV_QUAL[C/R]:30.86/25.25
T_STRAND_COUNTS[C/C/R/R]:2/7/2/10  \
                GERMLINE  UTR TPRG1L
chr1    3778838 3778838 +A      N_OBS_COUNTS[C/A/T]:5/7/22      N_AV_MM[C/R]:5.00/5.20
N_AV_MAPQ[C/R]:54.20/81.20  \
                N_NQS_MM_RATE[C/R]:0.00/0.01    N_NQS_AV_QUAL[C/R]:24.94/26.05
N_STRAND_COUNTS[C/C/R/R]:4/1/15/0  \
                T_OBS_COUNTS[C/A/T]:15/15/48    T_AV_MM[C/R]:9.73/4.21
T_AV_MAPQ[C/R]:91.53/86.09  \
                T_NQS_MM_RATE[C/R]:0.17/0.02    T_NQS_AV_QUAL[C/R]:30.57/25.19
T_STRAND_COUNTS[C/C/R/R]:15/0/32/1 \
                GERMLINE   INTRON  DFFB
```

The fields are tab-separated. The first four fields confer the same event and location information as in the brief format (chromosome, last reference base before the event, last reference base of the event, event itself). Event information is followed by tagged fields reporting various collected statistics. In the paired mode (as in the

example shown above), there will be two sets of the same statistics, one for normal (prefixed with 'N_') and one for tumor (prefixed with 'T_') samples. In the single sample mode, there will be only one set of statistics (for the only sample analyzed) and no 'N_'/'T_' prefixes. Statistics are stratified into (two or more of) the following classes: (C)onsensus-supporting reads (i.e. the reads that contain the called event, for which the line is printed); (A)ll reads that contain an indel at the site (not necessarily the called consensus); (R)eference allele-supporting reads, (T)otal=all reads.

For instance, the field T_OBS_COUNTS[C/A/T]:3/3/7 in the first line of the example above should be interpreted as follows: a) this is the OBS_COUNTS statistics for the (T)umor sample (this particular one is simply the read counts, all statistics are listed below); b) The statistics is broken down into three classes: [C/A/T]=(C)onsensus/(A)ll-indel/(T)otal coverage; c) the respective values in each class are 3, 3, 7. In other words, the insertion +G is observed in 3 distinct reads, there was a total of 3 reads with an indel at the site (i.e. only consensus was observed in this case with no observations for any other indel event), and the total coverage at the site is 7. Examining the N_OBS_COUNTS field in the same record, we can conclude that the total coverage in normal at the same site was 52, and among those reads there was not a single one carrying any indel (C/A/T=0/0/52). Hence the 'SOMATIC' annotation added towards the end of the line.

In paired mode the tagged statistics fields are always followed by GERMLINE/SOMATIC annotation (in single sample mode this field is skipped). If --refseq option is used, the next field will contain the coding status annotation (one of GENOMIC/INTRON/UTR/CODING), optionally followed by the gene name (present if the indel is within the boundaries of an annotated gene, i.e. the status is not GENOMIC).

List of annotations produced in verbose mode

NOTE: in older versions the OBS_COUNTS statistics was erroneously annotated as [C/A/R] (last class R, not T). This was a typo, and the last number reported in the triplet was still total coverage.

Duplicated reads, reads with mapping quality 0, or reads coming from blacklisted lanes are not counted and do not contribute to any of the statistics.

When no reads are available in a class (e.g. the count of consensus indel-supporting reads in normal sample is 0), all the other statistics for that class (e.g. average mismatches per read, average base qualities in NQS window etc) will be set to 0. For some statistics (average number of mismatches) this artificial value can be "very good", for some others (average base quality) it's "very bad". Needless to say, all those zeroes reported for the classes with no reads should be ignored when attempting call filtering.

- OBS_COUNTS[C/A/T] Observed counts of reads supporting the consensus (called) indel, all indels (consensus + any others), and the total coverage at the site, respectively.

- AV_MM[C/R] Average numbers of mismatches across consensus indel- and reference allele-supporting reads.

- AV_MAPQ[C/R] Average mapping qualities (as reported in the input bam file) of consensus indel- and reference allele-supporting reads.

- NQS_MM_RATE[C/R] Mismatch rate in small (currently 5bp on each side) window around the indel in consensus indel- and reference allele-supporting reads. The rate is obtained as average across all bases

falling into the window, in all reads. Namely, if the sum of coverages from all the consensus-supporting reads, at every individual reference base in [indel start-5,indel start],[indel stop, indel_stop +5] intervals is, e.g. 100, and 5 of those covering bases are mismatches (regardless of what particular read they come from or whether they occur at the same or different positions), the NQS_MM_RATE[C] is 0.05. Note that this statistics was observed to behave very differently from AV_MM. The latter captures potential global problems with read-placement and/or overall read quality issues: when reads have too many mismatches, the alignments are problematic. Even if the vicinity of the indel is "clean" (low NQS_MM_RATE), high AV_MM indicates a potential problem (e.g. the reads could have come from a highly othologous pseudogene/gene copy that is not in the reference). On the other hand, even when AV_MM is low (especially for long reads), so that the overall placement of the reads seem to be reliable, NQS_MM_RATE may still be relatively high, indicating a potential local problem (few low quality/mismatching bases near the tip of the read, incorrect indel event etc).

- NQS_AV_QUAL[C/R] Average base quality computed across all bases falling into the 5bp window on each side of the indel and coming form all consensus- or reference-supporting reads, respectively.

- STRAND_COUNTS[C/C/R/R] Counts of consensus-supporting forward aligned, consensus-supporting rc-aligned, reference-supporting forward-aligned and reference-supporting rc-aligned reads, respectively.

Creating a indel mask file

The output of the Somatic Indel Detector can be used to mask out SNPs near indels. To do this, we have a script that creates a bed file representing the masking intervals based on the output of this tool. Note that this script requires a full SVN checkout of the GATK, although the strategy is simple: for each indel, create an interval which extends N bases to either side of it.

```
python python/makeIndelMask.py <raw_indels> <mask_window> <output>
e.g.
python python/makeIndelMask.py indels.raw.bed 10 indels.mask.bed
```

Using the Unified Genotyper #1237

Last updated on 2013-02-22 17:26:27

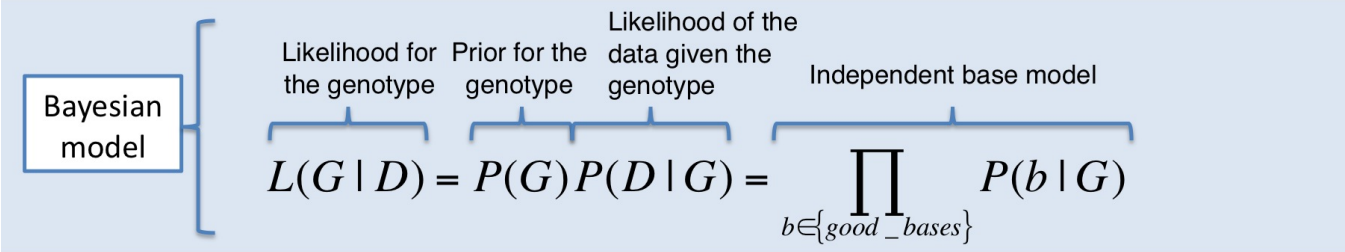
For a complete, detailed argument reference, refer to the technical documentation page.

1. Slides

The GATK requires the reference sequence in a single reference sequence in FASTA format, with all contigs in the same file. The GATK requires strict adherence to the FASTA standard. Only the standard ACGT bases are accepted; no non-standard bases (W, for example) are tolerated. Gzipped fasta files will not work with the GATK, so please make sure to unzip them first. Please see [Preparing the essential GATK input files: the reference genome] for more information on preparing FASTA reference sequences for use with the GATK.

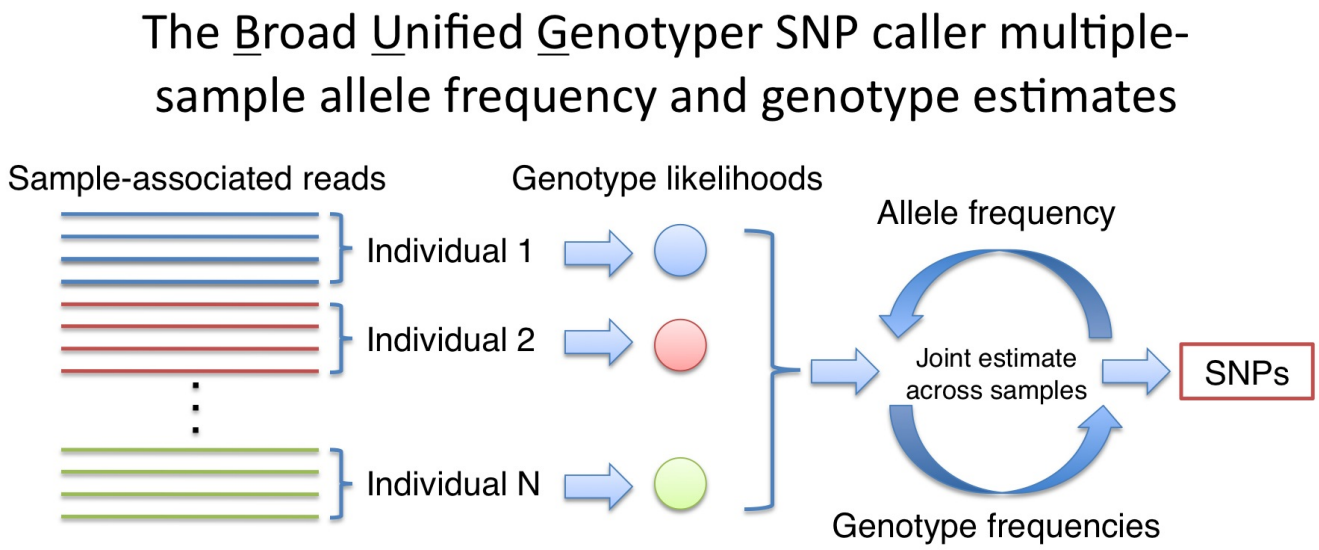
Genotype likelihoods

GATK single sample genotype likelihoods



- Priors applied during multi-sample calculation; P(G) = 1
- Likelihood of data computed using pileup of bases and associated quality scores at given locus
- Only “good bases” are included: those satisfying minimum base quality, mapping read quality, pair mapping quality, NQS
- P(b | G) uses a platform-specific confusion matrix
- L(G|D) computed for all 10 genotypes

Multiple-sample allele frequency and genotype estimates



- This approach allows us to combine weak single sample calls to discover variation among samples with high confidence

See http://www.broadinstitute.org/gsa/wiki/index.php/Unified_genotyper for more information

2. Relatively Recent Changes

The Unified Genotyper now makes multi-allelic variant calls!

Fragment-based calling

The Unified Genotyper calls SNPs via a two-stage inference, first from the reads to the sequenced fragments, and then from these inferred fragments to the chromosomal sequence of the organism. This two-stage system properly handles the correlation of errors between read pairs when the sequenced fragments contains errors itself. See Fragment-based calling PDF for more details and analysis.

The Allele Frequency Calculation

The allele frequency calculation model used by the Unified Genotyper computes a mathematically precise estimation of the allele frequency at a site given the read data. The mathematical derivation is similar to the one used by Samtools' `mpileup` tool. Heng Li has graciously allowed us to post the mathematical calculations backing the EXACT model here. Note that the calculations in the provided document assume just a single alternate allele for simplicity, whereas the Unified Genotyper has been extended to handle genotyping multi-allelic events. A slide showing the mathematical details for multi-allelic calling is available here.

3. Indel Calling with the Unified Genotyper

While the indel calling capabilities of the Unified Genotyper are still under active development, they are now in a stable state and are supported for use by external users. Please note that, as with SNPs, the Unified Genotyper is fairly aggressive in making a call and, consequently, the false positive rate will be high in the raw call set. We expect users to properly filter these results as per our best practices (which will be changing continually).

Note also that it is critical for the correct operation of the indel calling that the BAM file to be called is previously indel-realigned (see the IndelRealigner section on details). We strongly recommend doing joint Smith-Waterman alignment and not only per-lane or per-sample alignment at known sites. This is important because the caller is only empowered to genotype indels which are already present in reads.

Finally, while many of the parameters are common between indel and SNP calling, some parameters have different meaning or operate differently. For example, `--min_base_quality_score` has a fixed, well defined operation for SNPs (bases at a particular location with base quality lower than this threshold are ignored). However, indel calling is by definition delocalized and haplotype-based, so this parameter does not make sense. Instead, the indel caller will clip both ends of the reads if their quality is below a certain threshold (Q20), up to the point where there is a base in the read exceeding this threshold.

4. Miscellaneous notes

Note that the Unified Genotyper will not call indels in 454 data!

It's common to want to operate only over a part of the genome and to output SNP calls to standard output, rather than a file. The `-L` option lets you specify the region to process. If you set `-o` to `/dev/stdout` (or leave it out completely), output will be sent to the standard output of the console.

You can turn off logging completely by setting `-l OFF` so that the GATK operates in silent mode.

By default the Unified Genotyper downsamples each sample's coverage to no more than 250x (so there will be at most 250 * number_of_samples reads at a site). Unless there is a good reason for wanting to change this value, we suggest using this default value especially for exome processing; allowing too much coverage will require a lot more memory to run. When running on projects with many samples at low coverage (e.g. 1000 Genomes with 4x coverage per sample) we usually lower this value to about 10 times the average coverage: `-dcov 40`.

The Unified Genotyper does not use reads with a mapping quality of 255 ("unknown quality" according to the SAM specification). This filtering is enforced because the genotyper caps a base's quality by the mapping quality of its read (since the probability of the base's being correct depends on both qualities). We rely on sensible values for the mapping quality and therefore using reads with a 255 mapping quality is dangerous.

- That being said, if you are working with a data type where alignment quality cannot be determined, there is a (completely unsupported) workaround: the ReassignMappingQuality filter enables you to reassign the mapping quality of all reads on the fly. For example, adding `-rf ReassignMappingQuality -DMQ 60` to your command-line would change all mapping qualities in your bam to 60.

- Or, if you are working with data from a program like TopHat which uses MAPQ 255 to convey meaningful information, you can use the ReassignOneMappingQuality filter (new in 2.4) to assign a different MAPQ

value to those reads so they won't be ignored by GATK tools. For example, adding `-rf ReassignOneMappingQuality -RMQF 255 -RMQT 60` would change the mapping qualities of reads with MAPQ 255 in your bam to MAPQ 60.

5. Explanation of callable base counts

At the end of a GATK UG run, you should see if you have -l INFO enabled a report that looks like:

```
INFO  00:23:29,795 UnifiedGenotyper - Visited bases
247249719
INFO  00:23:29,796 UnifiedGenotyper - Callable bases
219998386
INFO  00:23:29,796 UnifiedGenotyper - Confidently called bases
219936125
INFO  00:23:29,796 UnifiedGenotyper - % callable bases of all loci
88.978
INFO  00:23:29,797 UnifiedGenotyper - % confidently called bases of all loci
88.953
INFO  00:23:29,797 UnifiedGenotyper - % confidently called bases of callable loci
88.953
INFO  00:23:29,797 UnifiedGenotyper - Actual calls made
303126
```

This is what these lines mean:

- **Visited bases**

This the total number of reference bases that were visited.

- **Callable bases**

Visited bases minus reference Ns and places with no coverage, which we never try to call.

- **Confidently called bases**

Callable bases that exceed the emit confidence threshold, either for being non-reference or reference. That is, if T is the min confidence, this is the count of bases where `QUAL > T` for the site being reference in all samples and/or `QUAL > T` for the site being non-reference in at least one sample.

Note a subtle implication of the last statement, with all samples vs. any sample: calling multiple samples tends to reduce the percentage of confidently callable bases, as in order to be confidently reference one has to be able to establish that **all** samples are reference, which is hard because of the stochastic coverage drops in each sample.

Note also that confidently called bases will rise with additional data per sample, so if you don't dedup your reads, include lots of poorly mapped reads, the numbers will increase. Of course, just because you confidently call the site doesn't mean that the data processing resulted in high-quality output, just that we had sufficient statistical

evident based on your input data to called ref / non-ref.

6. Calling sex chromosomes

The GATK can be used to call the sex (X and Y) chromosomes, without explicit knowledge of the gender of the samples. In an ideal world, with perfect upfront data processing, we would get perfect genotypes on the sex chromosomes without knowledge of who is diploid on X and has no Y, and who is hemizygous on both. However, misalignment and mismapping contributes especially to these chromosomes, as their reference sequence is clearly of lower quality than the autosomal regions of the genome. Nevertheless, it is possible to get reasonably good SNP calls, even with simple data processing and basic filtering. Results with proper, full data processing as per the best practices in the GATK should lead to very good calls. You can view a presentation "The GATK Unified Genotyper on chrX and chrY" in the GSA Public Drop Box.

Our general approach to calling on X and Y is to treat them just as we do the autosomes and then applying a gender-aware tools to correct the genotypes afterwards. It makes sense to filter out sites across all samples (outside PAR) that appear as confidently het in males, as well as sites on Y that appear confidently non-reference in females. Finally, it's possible to simply truncate the genotype likelihoods for males and females as appropriate from their diploid likelihoods -- AA, AB, and BB -- to their haploid equivalents -- AA and BB -- and adjust the genotype calls to reflect only these two options. We applied this approach in 1000G, but we only did it as the data went into imputation, so there's no simple tool to do this, unfortunately. The GATK team is quite interested in a general sex correction tool (analogous to the PhaseByTransmission tool for trios), so please do contact us if you are interested in contributing such a tool to the GATK codebase.

7. Related materials

- Explanation of the VCF Output

See Understanding the Unified Genotyper's VCF files.

Using the reference confidence calculation mode & generating a GVCF #2940

Last updated on 2013-09-10 20:47:55

As of GATK 2.7, the HaplotypeCaller is now capable of emitting a per-bp or summarized confidence estimate for a site being strictly homozygous-reference.

How it works

The mode works by assembling the reads to create potential haplotypes, realigning the reads to their most likely haplotypes, and then projecting these reads back onto the reference sequence via their haplotypes to compute alignments of the reads to the reference. For each position in the genome we have either a non-reference call (via the standard calling mechanism) or we can estimate the chance that some (unknown) non-reference allele is segregating at this position by examining the realigned reads that span the reference base. At this base we perform two calculations:

- Estimate the confidence that no SNP exists at the site by contrasting all reads with the ref base vs all reads with any non-reference base.

- Estimate the confidence that no indel of size < X (determined by command line parameter) could exist at this site by calculating the number of reads that provide evidence against such an indel, and from this value estimate the chance that we would not have seen the allele confidently.

Based on this, we emit the genotype likelihoods (PL) and compute the GQ (from the PLs) for the least confidence of these two models.

We use a symbolic allele, <NON_REF>, to indicate that the site is homozygous reference, and because we have an ALT allele we can provide allele-specific AD and PL field values.

How to use it

This mode can be enabled with the command line argument `--emitRefConfidence (-ERC)`, which has 3 options:

- NONE: don't emit anything (default)
- BP_RESOLUTION: emit detailed information for each BP
- GVCF: emit a block summarized version of the BP_RESOLUTION data

BP_RESOLUTION mode

This mode emits a very detailed model for the reference confidence at each reference base within the requested calling intervals. The output looks like:

```
20      10000435        .       T       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:54,0:53:54:99:0,160,2385
20      10000436        .       G       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:53,0:53:53:99:0,159,2011
20      10000437        .       C       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:53,0:53:53:99:0,159,1971
20      10000438        .       C       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:53,0:53:53:99:0,159,1816
20      10000439        .       T       G       1553.77 .
AC=2;AF=1.00;AN=2;DP=56;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=221.40;MQ0=0;QD=27.75
GT:AD:DP:GQ:PL  1/1:0,55:55:99:1582,164,0
20      10000440        .       T       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:55,0:55:55:99:0,166,2063
20      10000441        .       T       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:55,0:55:55:99:0,165,2000
20      10000442        .       T       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:56,0:56:56:99:0,168,2095
20      10000443        .       A       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:56,0:56:56:99:0,169,2089
20      10000444        .       A       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:56,0:56:56:99:0,168,2093
20      10000445        .       C       <NON_REF>       .       .       .
GT:AD:CD:DP:GQ:PL       0/0:56,0:56:56:99:0,169,2137
```

This shows each reference base in a 11 bp interval. Only one site (10000439) contains a SNP call. The other 10 bases are reference. For each reference base not part of a variant call we emit a VCF record with have the special symbolic allele <NON_REF> indicating the call is between the reference base and any possible non-reference allele that might be segregating at this site. The genotype fields include the standard GATK GT, AD, DP, GQ, and PL fields, all calculated as ref vs. any non-reference base. There's a new special CD field as well, that gives the number of indel informative reads at this location, given the maximum size of the indel to consider.

Note that there's no site-level QUAL field value. We discussed this internally and since the QUAL is the probability that the site is polymorphic, all of the QUAL field values should be 0 here, so we decided to drop it.

GVCF mode

GVCF mode summarizes the output of BP_RESOLUTION by grouping sequential reference calls with similar GQ values into blocks, and emitting summarized blocks instead of the values at each base pair. The blocking is discrete, deterministic, and can be controlled by the command line parameter GVCFGQBands. The algorithm creates bands of GQ values and merges sequential blocks with GQ values within the same band into a single synthetic block.

For example, if the GVCFBQBands argument is -GQB 10 -GQB 20 then we'll create blocks in the GVCF where sequential <NON_REF> calls have GQ values either < 10, between 10 and 19, or are 20 or greater.

The GVCF INFO and FORMAT fields have a few new annotations. First, END and BLOCK_SIZE indicate the extent of the reference block and its size (redundant yes but easy to read). For each GVCF sample field there's now only GT, DP and GQ from the standard annotations. GT is always 0/0 while DP and GQ are representative values (currently median) from the bp resolution DP and GQ values within the block. The MIN_DP and MIN_GQ are the minimum values seen within the block for their respective fields.

The output for the above example in GVCF mode looks like:

```
20      10000435        .       T       <NON_REF>       .       .
BLOCK_SIZE=4;END=10000438       GT:DP:GQ:MIN_DP:MIN_GQ  0/0:53:99:53:159
20      10000439        .       T       G       1553.77 .
AC=2;AF=1.00;AN=2;DP=56;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=221.40;MQ0=0;QD=27.75
GT:AD:DP:GQ:PL  1/1:0,55:55:99:1582,164,0
20      10000440        .       T       <NON_REF>       .       .
BLOCK_SIZE=6;END=10000445       GT:DP:GQ:MIN_DP:MIN_GQ  0/0:56:99:55:165
```

Here's a GVCF output for 100 bp around the same variant:

```
20      10000400        .       T       <NON_REF>       .       .
BLOCK_SIZE=39;END=10000438      GT:DP:GQ:MIN_DP:MIN_GQ  0/0:57:99:53:128
20      10000439        .       T       G       1553.77 .
AC=2;AF=1.00;AN=2;DP=56;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=221.40;MQ0=0;QD=27.75
GT:AD:DP:GQ:PL  1/1:0,55:55:99:1582,164,0
20      10000440        .       T       <NON_REF>       .       .
```

```
BLOCK_SIZE=61;END=10000500      GT:DP:GQ:MIN_DP:MIN_GQ  0/0:54:99:49:136
```

Current caveats

- Currently only works for single sample calling
- Currently slows down the calculation significantly (but this will be improved in the future)
- Only GVCF mode is likely to work with downstream GATK tools right now, as the special <NON_REF> allele may be treated as a polymorphic site. If you want to use the BP resolution mode you will likely need to split the VCF into the ref sites and polymorphic sites, perform standard filtering etc to the polymorphic VCF, and then merge the ref and non-ref VCFs back together

Feedback

We are interested in community feedback on the output format and additional information we should provide to make the reference model as useful as possible. Please post suggestions as comments here.

Variant Quality Score Recalibration (VQSR) #39

Last updated on 2013-10-03 13:55:34

This document describes what Variant Quality Score Recalibration (VQSR) is designed to do, and outlines how it works under the hood. For command-line examples and recommendations on what specific resource datasets and arguments to use for VQSR, please see this FAQ article.

Slides that explain the VQSR methodology in more detail as well as the individual component variant annotations can be found here in the GSA Public Drop Box.

Detailed information about command line options for VariantRecalibrator can be found here.

Detailed information about command line options for ApplyRecalibration can be found here.

Introduction

The purpose of the variant recalibrator is to assign a well-calibrated probability to each variant call in a call set. One can then create highly accurate call sets by filtering based on this single estimate for the accuracy of each call.

The approach taken by variant quality score recalibration is to develop a continuous, covarying estimate of the relationship between SNP call annotations (QD, SB, HaplotypeScore, HRun, for example) and the the probability that a SNP is a true genetic variant versus a sequencing or data processing artifact. This model is determined adaptively based on "true sites" provided as input, typically HapMap 3 sites and those sites found to be polymorphic on the Omni 2.5M SNP chip array. This adaptive error model can then be applied to both known and novel variation discovered in the call set of interest to evaluate the probability that each call is real. The score that gets added to the INFO field of each variant is called the VQSLOD. It is the log odds ratio of being a true variant versus being false under the trained Gaussian mixture model. The variant recalibrator contrastively

evaluates variants in a two step process, each performed by a distinct tool:

- VariantRecalibrator
  Create a Gaussian mixture model by looking at the annotations values over a high quality subset of the input call set and then evaluate all input variants.

- ApplyRecalibration
  Apply the model parameters to each variant in input VCF files producing a recalibrated VCF file in which each variant is annotated with its VQSLOD value. In addition, this step will filter the calls based on this new lod score by adding lines to the FILTER column for variants that don't meet the lod threshold as provided by the user (with the ts_filter_level parameter).

Recalibration tutorial with example HiSeq, single sample, deep coverage, whole genome call set

This brief tutorial uses example sequencing data produced at the Broad Institute to explain how you can use the variant quality score recalibrator and evaluate its performance. All of the data used in this tutorial is available in VCF format from our GATK resource bundle.

Note that this example is not updated with the latest Best Practices recommendations. For the latest recommendations and command lines, please see the FAQ article referenced above.

Input call set

input: NA12878.HiSeq.WGS.bwa.cleaned.raw.b37.subset.vcf

- These calls were generated with the UnifiedGenotyper from a 30X coverage modern, single sample run of HiSeq. They were randomly downsampled to keep the file size small but in general one would want to use the full set of variants available genome-wide for this procedure. No other pre-filtering steps were applied to the raw output.

Training sets

HapMap 3.3: hapmap_3.3.b37.sites.vcf

- These high quality sites are used both to train the Gaussian mixture model and then again when choosing a LOD threshold based on sensitivity to truth sites.

- The parameters for these sites will be: **known = false, training = true, truth = true, prior = Q15 (96.84%)**

Omni 2.5M chip: 1000G_omni2.5.b37.sites.vcf

- These polymorphic sites from the Omni genotyping array are used when training the model.

- The parameters for these sites will be: **known = false, training = true, truth = false, prior = Q12 (93.69%)**

dbSNP build 132: dbsnp_132.b37.vcf

- The dbsnp sites are generally considered to be not of high enough quality to be used in training but here we stratify output metrics such as ti/tv ratio by presence in dbsnp (known sites) or not (novel sites).

- The parameters for these sites will be: **known = true, training = false, truth = false, prior = Q8 (84.15%)**

The default prior for all other variants is **Q2 (36.90%)**. This low value reflects the fact that the philosophy of the UnifiedGenotyper is to produce a large, highly sensitive callset that needs to be heavily refined through additional filtering.

VariantRecalibrator

Detailed information about command line options for VariantRecalibrator can be found here.

Build a Gaussian mixture model using a high quality subset of the input variants and evaluate those model parameters over the full call set. The following notes describe the appropriate inputs to use for this tool.

- Note that this walker expects call sets in which each record has been appropriately annotated (see e.g. VariantAnnotator). Input call set rod bindings must start with "input". See the command line below.

- When constructing an initial call set (see e.g. Unified Genotyper or Haplotype Caller) for use with the Recalibrator, it's generally best to turn down the confidence threshold to allow more borderline calls (trusting the Recalibrator to keep the real ones while filtering out the false positives). For example, we often use a Q20 threshold on our deep coverage calls with the Recalibrator (whereas the default threshold in the UnifiedGenotyper is Q30).

- No pre-filtering is necessary when using the Recalibrator. See below for the advanced options which allow the user to selectively ignore certain filters if they have already been applied to your call set.

- The tool accepts any ROD bindings when specifying the set of truth sites to be used during modeling. Information about how to download VCF files which we routinely use for training is in the FAQ section at the bottom of the page.

- Each training set ROD binding is specified with key-value tags to qualify whether the set should be considered as known sites, training sites, and/or truth sites. Additionally, the prior probability of being true for those sites is also specified via these tags in Phred scale. See the command line below for an example. An explanation for how each of the training sets is used by the algorithm:

- Training sites: Input variants which are found to overlap with these training sites are used to build the Gaussian mixture model.

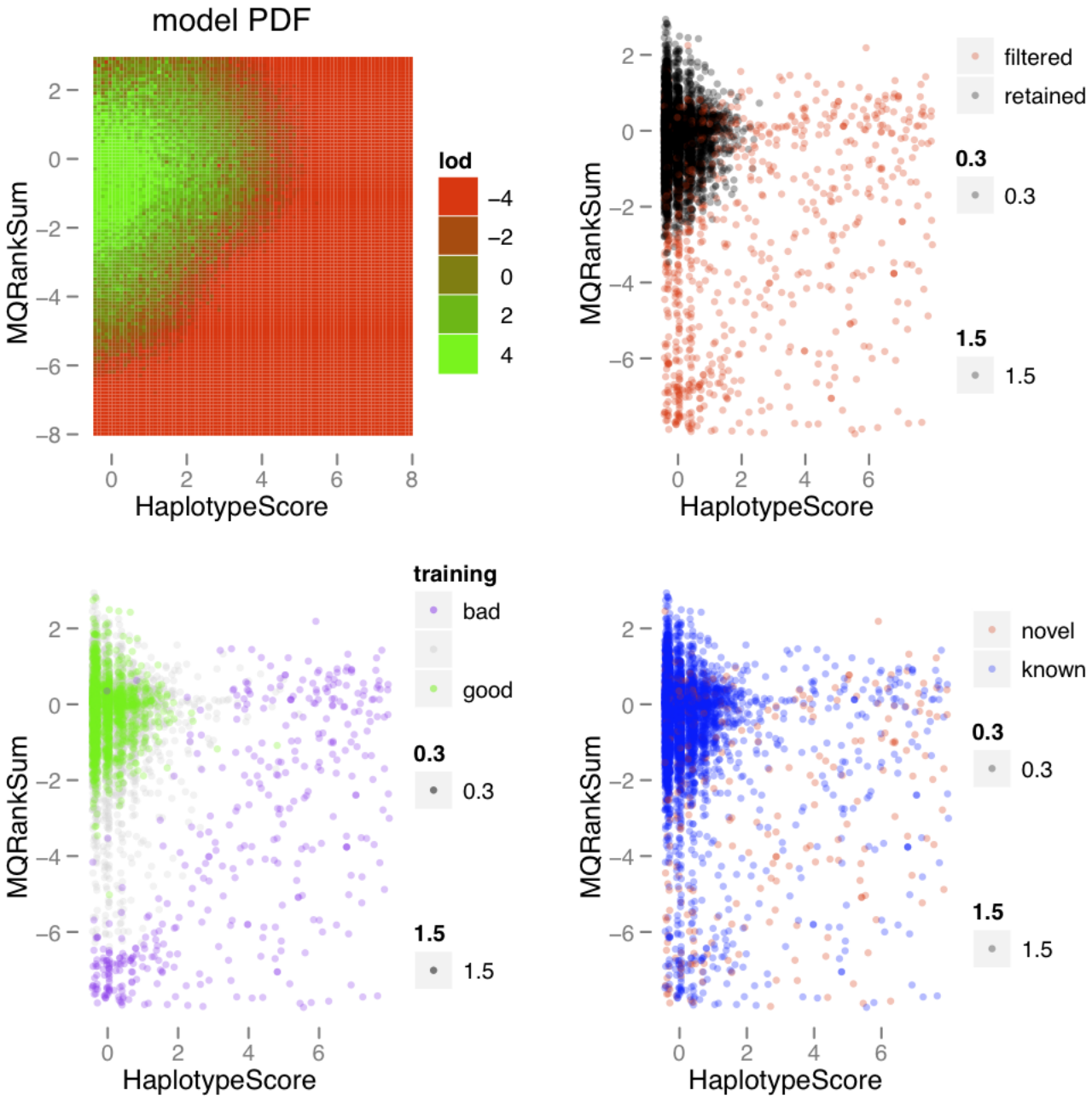
- Truth sites: When deciding where to set the cutoff in VQSLOD sensitivity to these truth sites is used. Typically one might want to say I dropped my threshold until I got back 99% of HapMap sites, for example.

- Known sites: The known / novel status of a variant isn't used by the algorithm itself and is only used for reporting / display purposes. The output metrics are stratified by known status in order to aid in comparisons with other call sets.

Interpretation of the Gaussian mixture model plots

The variant recalibration step fits a Gaussian mixture model to the contextual annotations given to each variant. By fitting this probability model to the training variants (variants considered to be true-positives), a probability can be assigned to the putative novel variants (some of which will be true-positives, some of which will be false-positives). It is useful for users to see how the probability model was fit to their data. Therefore a modeling report is automatically generated each time VariantRecalibrator is run (in the above command line the report will appear as path/to/output.plots.R.pdf). For every pair-wise combination of annotations used in modeling, a 2D

projection of the Gaussian mixture model is shown.



Gaussian mixture model report that is automatically generated by the VQSR from the example HiSeq call set. This page shows the 2D projection of mapping quality rank sum test versus Haplotype score by marginalizing over the other annotation dimensions in the model.

In each page there are four panels which show different ways of looking at the 2D projection of the model. The upper left panel shows the probability density function that was fit to the data. The 2D projection was created by marginalizing over the other annotation dimensions in the model via random sampling. Green areas show locations in the space that are indicative of being high quality while red areas show the lowest probability areas. In general putative SNPs that fall in the red regions will be filtered out of the recalibrated call set.

The remaining three panels give scatter plots in which each SNP is plotted in the two annotation dimensions as points in a point cloud. The scale for each dimension is in normalized units. The data for the three panels is the same but the points are colored in different ways to highlight different aspects of the data. In the upper right panel SNPs are colored black and red to show which SNPs are retained and filtered, respectively, by applying the VQSR procedure. The red SNPs didn't meet the given truth sensitivity threshold and so are filtered out of the call set. The lower left panel colors SNPs green, grey, and purple to give a sense of the distribution of the variants used to train the model. The green SNPs are those which were found in the training sets passed into the VariantRecalibrator step, while the purple SNPs are those which were found to be furthest away from the learned Gaussians and thus given the lowest probability of being true. Finally, the lower right panel colors each SNP by their known/novel status with blue being the known SNPs and red being the novel SNPs. Here the idea is to see if the annotation dimensions provide a clear separation between the known SNPs (most of which are true) and the novel SNPs (most of which are false).

An example of good clustering for SNP calls from the tutorial dataset is shown to the right. The plot shows that the training data forms a distinct cluster at low values for each of the two statistics shown (haplotype score and mapping quality bias). As the SNPs fall off the distribution in either one or both of the dimensions they are assigned a lower probability (that is, move into the red region of the model's PDF) and are filtered out. This makes sense as not only do higher values of HaplotypeScore indicate a lower chance of the data being explained by only two haplotypes but also higher values for mapping quality bias indicate more evidence of bias between the reference bases and the alternative bases. The model has captured our intuition that this area of the distribution is highly enriched for machine artifacts and putative variants here should be filtered out!

Tranches and the tranche plot

The recalibrated variant quality score provides a continuous estimate of the probability that each variant is true, allowing one to partition the call sets into quality tranches. The first tranche is exceedingly specific but less sensitive, and each subsequent tranche in turn introduces additional true positive calls along with a growing number of false positive calls. Downstream applications can select in a principled way more specific or more sensitive call sets or incorporate directly the recalibrated quality scores to avoid entirely the need to analyze only a fixed subset of calls but rather weight individual variant calls by their probability of being real. An example tranche plot, automatically generated by the VariantRecalibator walker, is shown on the right.

BAD IMAGE FILE (JPEG)

Try http://cdn.vanillaforums.com/gatk.vanillaforums.com/FileUpload/b6/ef1c4b5fe263e3a24fea6848776cd8.jpeg

Tranches plot for example HiSeq call set. The x-axis gives the number of novel variants called while the y-axis shows two quality metrics -- novel transition to transversion ratio and the overall truth sensitivity.

Ti/Tv-free recalibration

We use a Ti/Tv-free approach to variant quality score recalibration. This approach requires an additional truth data set, and cuts the VQSLOD at given sensitivities to the truth set. It has several advantages over the Ti/Tv-targeted approach:

- The truth sensitivity (TS) approach gives you back the novel Ti/Tv as a QC metric [YES!]
- The truth sensitivity (TS) approach is conceptual cleaner than deciding on a novel Ti/Tv target for your dataset
- The TS approach is easier to explain and defend, as saying "I took called variants until I found 99% of my known variable sites" is easier than "I took variants until I dropped my novel Ti/Tv ratio to 2.07"

We have used hapmap 3.3 sites as the truth set (genotypes_r27_nr.b37_fwd.vcf), but other sets of high-quality (~99% truly variable in the population) sets of sites should work just as well. In our experience, with HapMap, 99% is a good threshold, as the remaining 1% of sites often exhibit unusual features like being close to indels or are actually MNPs, and so receive a low VQSLOD score.

Note that the expected Ti/Tv is still an available argument but it is only used for display purposes.

ApplyRecalibration

Detailed information about command line options for ApplyRecalibration can be found here.

Using the tranche file generated by the previous step the ApplyRecalibration walker looks at each variant's VQSLOD value and decides which tranche it falls in. Variants in tranches that fall below the specified truth sensitivity filter level have their filter field annotated with its tranche level. This will result in a call set that simultaneously is filtered to the desired level but also has the information necessary to pull out more variants at a slightly lower quality level.

Frequently Asked Questions

How do I know which annotations to use for my data?

The five annotation values provided in the command lines above (QD, HaplotypeScore, MQRankSum, ReadPosRankSum, and HRun) have been show to give good results for a variety of data types. However this shouldn't be taken to mean these annotations give the absolute best modeling for every source of sequencing data. Better results could possibly be achieved through experimentation with which SNP annotations are used in the algorithm. The goal is to find annotation values with are approximately Gaussianly distributed and also serve to separate the probably true (known) SNPs from the probably false (novel) SNPs.

How do I know which -tranche arguments to pass into the VariantRecalibrator step?

The -tranche arguments main purpose is to create the tranche plot (as shown above). They are meant to convey the idea that with real, calibrated variant quality scores one can create call sets in which each variant doesn't have to have a hard answer as to whether it is in or out of the set. If a very high accuracy call set is desired then one can use the highest tranche, but if a larger, more complete call set is a higher priority than one can dip down into lower and lower tranches. These tranches are applied to the output VCF file using the FILTER field. In this way an end user can choose to use some of the filtered records or only use the PASSing records. For new users to the variant quality score recalibrator perhaps the easiest thing to do in the beginning is simply select the single desired false discovery rate and pass that value in as a single -tranche argument to make sure that the desired rate can be achieved given the other parameters to the algorithm.

What should I use as training data?

See the FAQ article referenced above for the most up to date recommendations.

Generally speaking, the VariantRecalibrator step accept lists of truth and training sites in several formats (dbsnp ROD, VCF, and BED, for example). Any list can be used but it is best to use only those sets which are of the best quality. The truth sets are passed into the algorithm using any rod binding name and their truth or training status is specified with rod tags (see VariantRecalibrator section above). We routinely use the HapMap v3.3 VCF file and the Omni2.5M SNP chip array in training the model. In general the false positive rate of dbsnp sites is too high to be used reliably for training the model.

HapMap v3.3 as well as the Omni validation array VCF files are available in our GATK resource bundle.

Does the VQSR work with non-human variant calls?

Absolutely! The VQSR accepts any list of sites to use as training / truth data, not just HapMap.

Don't have any truth data for your organism? No problem. There are several things one might experiment with. One idea is to first do an initial round of SNP calling and only use those SNPs which have the highest quality scores. These sites which have the most confidence are probably real and could be used as truth data to help disambiguate the rest of the variants in the call set. Another idea is to try using several SNP callers in addition to the UnifiedGenotyper or HaplotypeCaller, and use those sites which are concordant between the different methods as truth data. There are many fruitful avenues of research here. Hopefully the model reporting plots help facilitate this experimentation.

Perhaps the best place to begin is to use a line like the following when specifying the truth set:

```
-resource:concordantSet,known=true,training=true,truth=true,prior=10.0
path/to/concordantSet.vcf
```

Can I use the variant quality score recalibrator with my small sequencing experiment?

This tool is expecting thousands of variant sites in order to achieve decent modeling with the Gaussian mixture model. Whole exome call sets work well, but anything smaller than that scale might run into difficulties.

One piece of advice is to turn down the number of Gaussians used during training and to turn up the number of variants that are used to train the negative model. This can be accomplished by adding `--maxGaussians 4 --numBadVariants 1000` to your command line.

`numBadVariants` is the number of variants that the program will use to build the negative model. If you have a small callset, you may need to use a larger amount in order to have enough "bad" variants to build the negative model. `maxGaussians` is the maximum number of different "clusters" (=Gaussians) of variants the program is "allowed" to try to identify. Lowering this number forces the program to group variants into a smaller number of clusters, which means there will be more variants in each cluster -- hopefully enough to satisfy the statistical requirements. Of course, this decreases the level of discrimination that you can achieve between variant profiles/error modes. It's all about trade-offs; and unfortunately if you don't have a lot of variants you can't afford to be very demanding in terms of resolution.

Why don't all the plots get generated for me?

The most common problem related to this is not having Rscript accessible in your environment path. Rscript is the command line version of R that gets installed right alongside. We also make use of the ggplot2 library so please be sure to install that package as well.

FAQs

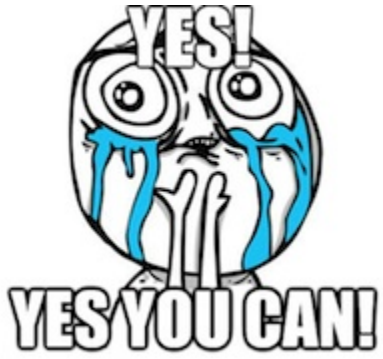
This section lists (and answers!) frequently asked questions.

These documentation articles cover specific points of clarification about the following:

- details of how the GATK tools work and how they should be applied to datasets
- questions that are related to NGS formats and concepts but are not specific to the GATK
- questions about the community forum, documentation website and user support system

Can I use GATK on non-diploid organisms? #1214

Last updated on 2013-09-13 17:44:19



In general most GATK tools don't care about ploidy. The major exception is, of course, at the variant calling step: the variant callers need to know what ploidy is assumed for a given sample in order to perform the appropriate calculations.

Since version 2.0, the UnifiedGenotyper has been able to deal with ploidies other than two. Three use cases are currently supported:

- Native variant calling in haploid or polyploid organisms.
- Pooled calling where many pooled organisms share a single barcode and hence are treated as a single "sample".
- Pooled validation/genotyping at known sites.

In order to enable this feature, you need to set the `-ploidy` argument to desired number of chromosomes per organism. In the case of pooled sequencing experiments, this argument should be set to the number of chromosomes per barcoded sample, i.e. `(Ploidy per individual) * (Individuals in pool)`.

Note that all other UnifiedGenotyper arguments work in the same way.

A full minimal command line would look for example like

```
java -jar GenomeAnalysisTK.jar \
     -R reference.fasta \
     -I myReads.bam \
     -T UnifiedGenotyper \
     -ploidy 4
```

The `glm` argument works in the same way as in the diploid case - set to `[INDEL|SNP|BOTH]` to specify which variants to discover and/or genotype.

Current Limitations

Many of these limitations will be gradually removed over time, but for now please keep these in mind.

- Fragment-aware calling like the one provided by default for diploid organisms is not present for the non-diploid case.
- Some annotations do not work in non-diploid cases. In particular, InbreedingCoeff will not be annotated on non-diploid calls. Annotations that do work and are supported in non-diploid use cases are the following: `QUAL`, `QD`, `SB`, `FS`, `AC`, `AF`, and Genotype annotations such as `PL`, `AD`, `GT`, etc.
- The HaplotypeCaller and ReduceReads currently do not support non-diploid data.
- In theory you can use VQSR to filter non-diploid calls, but we currently have no experience with this and therefore cannot offer any support nor best practices on how to do this.
- For indels, only a maximum of 4 alleles can be genotyped. This is not relevant for the SNP case, but discovering or genotyping more than this number of indel alleles will not work and an arbitrary set of 4 alleles will be chosen at a site.

You should also be aware of the fundamental accuracy limitations of high ploidy calling. Calling low-frequency variants in a pool or in an organism with high ploidy is hard because these rare variants become almost indistinguishable from sequencing errors.

Collected FAQs about BAM files #1317

Last updated on 2013-03-05 17:58:44

1. What file formats do you support for sequencer output?

The GATK supports the BAM format for reads, quality scores, alignments, and metadata (e.g. the lane of sequencing, center of origin, sample name, etc.). No other file formats are supported.

2. How do I get my data into BAM format?

The GATK doesn't have any tools for getting data into BAM format, but many other toolkits exist for this purpose. We recommend you look at Picard and Samtools for creating and manipulating BAM files. Also, many aligners are starting to emit BAM files directly. See BWA for one such aligner.

3. What are the formatting requirements for my BAM file(s)?

All BAM files must satisfy the following requirements:

- It must be aligned to one of the references described here.
- It must be sorted in coordinate order (not by queryname and not "unsorted").
- It must list the read groups with sample names in the header.
- Every read must belong to a read group.
- The BAM file must pass Picard validation.

See the BAM specification for more information.

4. What is the canonical ordering of human reference contigs in a BAM file?

It depends on whether you're using the NCBI/GRC build 36/build 37 version of the human genome, or the UCSC hg18/hg19 version of the human genome. While substantially equivalent, the naming conventions are different. The canonical ordering of contigs for these genomes is as follows:

Human genome reference consortium standard ordering and names (b3x): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, X, Y, MT...

UCSC convention (hg1x): chrM, chr1, chr2, chr3, chr4, chr5, chr6, chr7, chr8, chr9, chr10, chr11, chr12, chr13, chr14, chr15, chr16, chr17, chr18, chr19, chr20, chr21, chr22, chrX, chrY...

5. How can I tell if my BAM file is sorted properly?

The easiest way to do it is to download Samtools and run the following command to examine the header of your file:

```
$ samtools view -H /path/to/my.bam
@HD     VN:1.0  GO:none SO:coordinate
@SQ     SN:1    LN:247249719
@SQ     SN:2    LN:242951149
@SQ     SN:3    LN:199501827
@SQ     SN:4    LN:191273063
@SQ     SN:5    LN:180857866
@SQ     SN:6    LN:170899992
@SQ     SN:7    LN:158821424
@SQ     SN:8    LN:146274826
@SQ     SN:9    LN:140273252
@SQ     SN:10   LN:135374737
@SQ     SN:11   LN:134452384
@SQ     SN:12   LN:132349534
@SQ     SN:13   LN:114142980
@SQ     SN:14   LN:106368585
@SQ     SN:15   LN:100338915
@SQ     SN:16   LN:88827254
```

```
@SQ     SN:17   LN:78774742
@SQ     SN:18   LN:76117153
@SQ     SN:19   LN:63811651
@SQ     SN:20   LN:62435964
@SQ     SN:21   LN:46944323
@SQ     SN:22   LN:49691432
@SQ     SN:X    LN:154913754
@SQ     SN:Y    LN:57772954
@SQ     SN:MT   LN:16571
@SQ     SN:NT_113887    LN:3994
...
```

If the order of the contigs here matches the contig ordering specified above, and the `SO:coordinate` flag appears in your header, then your contig and read ordering satisfies the GATK requirements.

6. My BAM file isn't sorted that way. How can I fix it?

Picard offers a tool called SortSam that will sort a BAM file properly. A similar utility exists in Samtools, but we recommend the Picard tool because SortSam will also set a flag in the header that specifies that the file is correctly sorted, and this flag is necessary for the GATK to know it is safe to process the data. Also, you can use the ReorderSam command to make a BAM file SQ order match another reference sequence.

7. How can I tell if my BAM file has read group and sample information?

A quick Unix command using Samtools will do the trick:

```
$ samtools view -H /path/to/my.bam | grep '^@RG'
@RG ID:0    PL:solid
PU:Solid0044_20080829_1_Pilot1_Ceph_12414_B_lib_1_2Kb_MP_Pilot1_Ceph_12414_B_lib_1_2Kb_MP
LB:Lib1 PI:2750 DT:2008-08-28T20:00:00-0400 SM:NA12414  CN:bcm
@RG ID:1    PL:solid
PU:0083_BCM_20080719_1_Pilot1_Ceph_12414_B_lib_1_2Kb_MP_Pilot1_Ceph_12414_B_lib_1_2Kb_MP
LB:Lib1 PI:2750 DT:2008-07-18T20:00:00-0400 SM:NA12414  CN:bcm
@RG ID:2    PL:LS454    PU:R_2008_10_02_06_06_12_FLX01080312_retry  LB:HL#01_NA11881    PI:0
   SM:NA11881  CN:454MSC
@RG ID:3    PL:LS454    PU:R_2008_10_02_06_07_08_rig19_retry    LB:HL#01_NA11881    PI:0
SM:NA11881  CN:454MSC
@RG ID:4    PL:LS454    PU:R_2008_10_02_17_50_32_FLX03080339_retry  LB:HL#01_NA11881    PI:0
   SM:NA11881  CN:454MSC
...
```

The presence of the `@RG` tags indicate the presence of read groups. Each read group has a `SM` tag, indicating the sample from which the reads belonging to that read group originate.

In addition to the presence of a read group in the header, each read must belong to one and only one read group. Given the following example reads,

```
$ samtools view /path/to/my.bam | grep '^@RG'
EAS139_44:2:61:681:18781    35  1   1   0   51M =   9   59
TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA B<>;==?=?<==?=?=>>?>><=<?=?8<=?>?<:=?>?<
==?=>:;<?:= RG:Z:4  MF:i:18 Aq:i:0  NM:i:0  UQ:i:0  H0:i:85 H1:i:31
EAS139_44:7:84:1300:7601    35  1   1   0   51M =   12  62
TAACCCTAAGCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA G<>;==?=?&=>?=?<==?>?<>>?=?<==?>?<==?>
?1==@>?;<=><; RG:Z:3  MF:i:18 Aq:i:0  NM:i:1  UQ:i:5  H0:i:0  H1:i:85
EAS139_44:8:59:118:13881    35  1   1   0   51M =   2   52
TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA @<>;<=?=?==>?>?<==?=><=>?-?;=>?:><==?7?;
<>?5?<<=>:; RG:Z:1  MF:i:18 Aq:i:0  NM:i:0  UQ:i:0  H0:i:85 H1:i:31
EAS139_46:3:75:1326:2391    35  1   1   0   51M =   12  62
TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA @<>==>?>@???B>A>?>A?A>??A?@>?@A?@;??A>@7
>?>>@:>=@;@ RG:Z:0  MF:i:18 Aq:i:0  NM:i:0  UQ:i:0  H0:i:85 H1:i:31
...
```

membership in a read group is specified by the `RG:Z:*` tag. For instance, the first read belongs to read group 4 (sample NA11881), while the last read shown here belongs to read group 0 (sample NA12414).

8. My BAM file doesn't have read group and sample information. Do I really need it?

Yes! Many algorithms in the GATK need to know that certain reads were sequenced together on a specific lane, as they attempt to compensate for variability from one sequencing run to the next. Others need to know that the data represents not just one, but many samples. Without the read group and sample information, the GATK has no way of determining this critical information.

9. What's the meaning of the standard read group fields?

For technical details, see the SAM specification on the Samtools website.

Tag	Importance	SAM spec definition	Meaning
ID	Required	Read group identifier. Each @RG line must have a unique ID. The value of ID is used in the RG tags of alignment records. Must be unique among all read groups in header section. Read groupIDs may be modified when merging SAM files in order to handle collisions.	Ideally, this should be a globally unique identify across all sequencing data in the world, such as the Illumina flowcell + lane name and number. Will be referenced by each read with the RG:Z field, allowing tools to determine the read group information associated with each read, including the sample from which the read came. Also, a read group is effectively treated as a separate run of the NGS instrument in tools like base quality score recalibration -- all reads within a read group are assumed to come from the same instrument run and to therefore share the same error model.

SM	Sample. Use pool name where a pool is being sequenced.	Required. As important as ID.	The name of the sample sequenced in this read group. GATK tools treat all read groups with the same SM value as containing sequencing data for the same sample. Therefore it's critical that the SM field be correctly specified, especially when using multi-sample tools like the Unified Genotyper.
PL	Platformtechnology used to produce the read. Valid values: ILLUMINA, SOLID, LS454, HELICOS and PACBIO.	Important. Not currently used in the GATK, but was in the past, and may return. The only way to known the sequencing technology used to generate the sequencing data .	It's a good idea to use this field.
LB	DNA preparation library identify	Essential for MarkDuplicates	MarkDuplicates uses the LB field to determine which read groups might contain molecular duplicates, in case the same DNA library was sequenced on multiple lanes.

We do not require value for the `CN`, `DS`, `DT`, `PG`, `PI`, or `PU` fields.

A concrete example may be instructive. Suppose I have a trio of samples: MOM, DAD, and KID. Each has two DNA libraries prepared, one with 400 bp inserts and another with 200 bp inserts. Each of these libraries is run on two lanes of an Illumina HiSeq, requiring 3 x 2 x 2 = 12 lanes of data. When the data come off the sequencer, I would create 12 bam files, with the following `@RG` fields in the header:

```
Dad's data:
@RG     ID:FLOWCELL1.LANE1      PL:ILLUMINA     LB:LIB-DAD-1 SM:DAD      PI:200
@RG     ID:FLOWCELL1.LANE2      PL:ILLUMINA     LB:LIB-DAD-1 SM:DAD      PI:200
@RG     ID:FLOWCELL1.LANE3      PL:ILLUMINA     LB:LIB-DAD-2 SM:DAD      PI:400
@RG     ID:FLOWCELL1.LANE4      PL:ILLUMINA     LB:LIB-DAD-2 SM:DAD      PI:400

Mom's data:
@RG     ID:FLOWCELL1.LANE5      PL:ILLUMINA     LB:LIB-MOM-1 SM:MOM      PI:200
@RG     ID:FLOWCELL1.LANE6      PL:ILLUMINA     LB:LIB-MOM-1 SM:MOM      PI:200
@RG     ID:FLOWCELL1.LANE7      PL:ILLUMINA     LB:LIB-MOM-2 SM:MOM      PI:400
@RG     ID:FLOWCELL1.LANE8      PL:ILLUMINA     LB:LIB-MOM-2 SM:MOM      PI:400

Kid's data:
@RG     ID:FLOWCELL2.LANE1      PL:ILLUMINA     LB:LIB-KID-1 SM:KID      PI:200
@RG     ID:FLOWCELL2.LANE2      PL:ILLUMINA     LB:LIB-KID-1 SM:KID      PI:200
@RG     ID:FLOWCELL2.LANE3      PL:ILLUMINA     LB:LIB-KID-2 SM:KID      PI:400
@RG     ID:FLOWCELL2.LANE4      PL:ILLUMINA     LB:LIB-KID-2 SM:KID      PI:400
```

Note the hierarchical relationship between read groups (unique for each lane) to libraries (sequenced on two lanes) and samples (across four lanes, two lanes for each library).

9. My BAM file doesn't have read group and sample information. How do I add it?

Use Picard's AddOrReplaceReadGroups tool to add read group information.

10. How do I know if my BAM file is valid?

Picard contains a tool called ValidateSamFile that can be used for this. BAMs passing `STRICT` validation stringency work best with the GATK.

11. What's the best way to create a subset of my BAM file containing only reads over a small interval?

You can use the GATK to do the following:

```
GATK -I full.bam -T PrintReads -L chr1:10-20 -o subset.bam
```

and you'll get a BAM file containing only reads overlapping those points. This operation retains the complete BAM header from the full file (this was the reference aligned to, after all) so that the BAM remains easy to work with. We routinely use these features for testing and high-performance analysis with the GATK.

Collected FAQs about VCF files #1318

Last updated on 2012-10-18 15:00:51

1. What file formats do you support for variant callsets?

We support the Variant Call Format (VCF) for variant callsets. No other file formats are supported.

2. How can I know if my VCF file is valid?

VCFTools contains a validation tool that will allow you to verify it.

3. Are you planning to include any converters from different formats or allow different input formats than VCF?

No, we like VCF and we think it's important to have a good standard format. Multiplying formats just makes life hard for everyone, both developers and analysts.

Collected FAQs about interval lists #1319

Last updated on 2013-01-15 02:59:32

1. What file formats do you support for interval lists?

We support three types of interval lists, as mentioned here. Interval lists should preferentially be formatted as Picard-style interval lists, with an explicit sequence dictionary, as this prevents accidental misuse (e.g. hg18 intervals on an hg19 file). Note that this file is 1-based, not 0-based (first position in the genome is position 1).

2. I have two (or more) sequencing experiments with different target intervals. How can I combine them?

One relatively easy way to combine your intervals is to use the online tool Galaxy, using the `Get Data -> Upload` command to upload your intervals, and the `Operate on Genomic Intervals` command to compute the intersection or union of your intervals (depending on your needs).

How can I access the GSA public FTP server? #1215

Last updated on 2012-10-18 14:51:28

We make various files available for public download from the GSA FTP server, such as the GATK resource bundle and presentation slides. We also maintain a public upload feature for processing bug reports from users.

There are two logins to choose from depending on whether you want to upload or download something:

Downloading

```
location: ftp.broadinstitute.org
username: gsapubftp-anonymous
password: <blank>
```

Uploading

```
location: ftp.broadinstitute.org
username: gsapubftp
password: 5WvQWSfi
```

How can I filter my callset if I cannot use VQSR / recalibrate variants ? #3225

Last updated on 2013-09-11 21:56:01

If you are sure that you cannot use VQSR / recalibrate variants (typically because your dataset is too small, or because there are no truth/training resources available for your organism), then you will need to use the VariantFiltration tool to manually filter your variants. To do this, you will need to compose filter expressions as explained here, here and here based on the recommendations detailed further below.

But first, some caveats

Let's be painfully clear about this: there is no magic formula that will give you perfect results. Filtering variants manually, using thresholds on annotation values, is subject to all sorts of caveats. The appropriateness of both the annotations and the threshold values is very highly dependent on the specific callset, how it was called, what the data was like, etc.

HOWEVER, because we want to help and people always say that something is better than nothing (not necessarily true, but let's go with that for now), we have formulated some **generic recommendations** that should at least provide **a starting point for people to experiment with their data**.

In case you didn't catch that bit in bold there, we're saying that you absolutely SHOULD NOT expect to run these commands and be done with your analysis. You absolutely SHOULD expect to have to evaluate your results critically and TRY AGAIN with some parameter adjustments until you find the settings that are right for your data.

In addition, please note that these recommendations are mainly designed for dealing with very small data sets (in terms of both number of samples or size of targeted regions). If you are not using VQSR because you do not have training/truth resources available for your organism, then you should expect to have to do even more tweaking on the filtering parameters.

So, here are some recommended arguments to use with VariantFiltration when ALL other options are unavailable to you:

Filtering recommendations for SNPs:

- `QD < 2.0`
- `MQ < 40.0`
- `FS > 60.0`
- `HaplotypeScore > 13.0`
- `MQRankSum < -12.5`
- `ReadPosRankSum < -8.0`

Filtering recommendations for indels:

- `QD < 2.0`
- `ReadPosRankSum < -20.0`
- `InbreedingCoeff < -0.8`
- `FS > 200.0`

And now some more IMPORTANT caveats (don't skip this!)

- The InbreedingCoeff statistic is a population-level calculation that is only available with 10 or more samples. If you have fewer samples you will need to omit that particular filter statement.

- For shallow-coverage (<10x), it is virtually impossible to use manual filtering to reliably separate true positives from false positives. You really, really, really should use the protocol involving variant quality score recalibration. If you can't do that, maybe you need to take a long hard look at your experimental design. In any case you're probably in for a world of pain.

- The maximum DP (depth) filter only applies to whole genome data, where the probability of a site having exactly N reads given an average coverage of M is a well-behaved function. First principles suggest this should be a binomial sampling but in practice it is more a Gaussian distribution. Regardless, the DP threshold should be set a 5 or 6 sigma from the mean coverage across all samples, so that the DP > X threshold eliminates sites with excessive coverage caused by alignment artifacts. Note that **for exomes, a straight DP filter shouldn't be used** because the relationship between misalignments and depth isn't clear for capture data.

Finally, a note of hope

Some bits of this article may seem harsh, or depressing. Sorry. We believe in giving you the cold hard truth.

HOWEVER, we do understand that this is one of the major points of pain that GATK users encounter -- along with understanding how VQSR works, so really, whichever option you go with, you're going to suffer.

And we do genuinely want to help. So although we can't look at every single person's callset and give an opinion on how it looks (no, seriously, don't ask us to do that), we do want to hear from you about how we can best help you help yourself. What information do you feel would help you make informed decisions about how to set parameters? Are the meanings of the annotations not clear? Would knowing more about how they are computed help you understand how you can use them? Do you want more math? Less math, more concrete examples?

Tell us what you'd like to see here, and we'll do our best to make it happen. (no unicorns though, we're out of stock)

We also welcome testimonials from you. We are one small team; you are a legion of analysts all trying different things. Please feel free to come forward and share your findings on what works particularly well in your hands.

How can I pass read filters and their arguments at the command line? #2338

Posted on 2013-03-13 15:26:50

Most GATK tools apply several read filters by default. You can look up exactly what are the defaults for each tool in their respective Technical Documentation pages.

But sometimes you want to specify additional filters yourself (and before you ask, no, you cannot disable the default read filters used by a given tool). This is how you do it:

The `--read-filter` argument (or `-rf` for short) allows you to apply whatever read filters you'd like. For example, to add the `MaxReadLengthFilter` filter above to `PrintReads`, you just add this to your command line:

```
--read_filter MaxReadLength
```

Notice that when you specify a read filter, you need to strip the Filter part of its name off!

The read filter will be applied with its default value (which you can also look up in the Tech Docs for that filter). Now, if you want to specify a different value from the default, you pass the relevant argument by adding this right after the read filter:

```
--read_filter MaxReadLength -maxReadLength 76
```

It's important that you pass the argument right after the filter itself, otherwise the command line parser won't know that they're supposed to go together.

And of course, you can add as many filters as you like by using multiple copies of the `--read_filter`

parameter:

```
--read_filter MaxReadLength --maxReadLength 76 --read_filter ZeroMappingQualityRead
```

How can I prepare a FASTA file to use as reference? #1601

Last updated on 2013-09-13 17:29:43

This article describes the steps necessary to prepare your reference file (if it's not one that you got from us). As a complement to this article, see the relevant tutorial.

Why these steps are necessary

The GATK uses two files to access and safety check access to the reference files: a `.dict` dictionary of the contig names and sizes and a `.fai` fasta index file to allow efficient random access to the reference bases. You have to generate these files in order to be able to use a Fasta file as reference.

NOTE: Picard and samtools treat spaces in contig names differently. We recommend that you avoid using spaces in contig names.

Creating the fasta sequence dictionary file

We use CreateSequenceDictionary.jar from Picard to create a .dict file from a fasta file.

```
> java -jar CreateSequenceDictionary.jar R= Homo_sapiens_assembly18.fasta O=
Homo_sapiens_assembly18.dict
[Fri Jun 19 14:09:11 EDT 2009] net.sf.picard.sam.CreateSequenceDictionary R=
Homo_sapiens_assembly18.fasta O= Homo_sapiens_assembly18.dict
[Fri Jun 19 14:09:58 EDT 2009] net.sf.picard.sam.CreateSequenceDictionary done.
Runtime.totalMemory()=2112487424
44.922u 2.308s 0:47.09 100.2%   0+0k 0+0io 2pf+0w
```

This produces a SAM-style header file describing the contents of our fasta file.

```
> cat Homo_sapiens_assembly18.dict
@HD     VN:1.0  SO:unsorted
@SQ     SN:chrM LN:16571
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:d2ed829b8a1628d16cbeee88e88e39eb
@SQ     SN:chr1 LN:247249719
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:9ebc6df9496613f373e73396d5b3b6b6
@SQ     SN:chr2 LN:242951149
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:b12c7373e3882120332983be99aeb18d
@SQ     SN:chr3 LN:199501827
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
```

```
   M5:0e48ed7f305877f66e6fd4addbae2b9a
@SQ     SN:chr4 LN:191273063
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:cf37020337904229dca8401907b626c2
@SQ     SN:chr5 LN:180857866
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:031c851664e31b2c17337fd6f9004858
@SQ     SN:chr6 LN:170899992
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:bfe8005c536131276d448ead33f1b583
@SQ     SN:chr7 LN:158821424
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:74239c5ceee3b28f0038123d958114cb
@SQ     SN:chr8 LN:146274826
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:1eb00fe1ce26ce6701d2cd75c35b5ccb
@SQ     SN:chr9 LN:140273252
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:ea244473e525dde0393d353ef94f974b
@SQ     SN:chr10        LN:135374737
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:4ca41bf2d7d33578d2cd7ee9411e1533
@SQ     SN:chr11        LN:134452384
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:425ba5eb6c95b60bafbf2874493a56c3
@SQ     SN:chr12        LN:132349534
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:d17d70060c56b4578fa570117bf19716
@SQ     SN:chr13        LN:114142980
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:c4f3084a20380a373bbbdb9ae30da587
@SQ     SN:chr14        LN:106368585
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:c1ff5d44683831e9c7c1db23f93fbb45
@SQ     SN:chr15        LN:100338915
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:5cd9622c459fe0a276b27f6ac06116d8
@SQ     SN:chr16        LN:88827254
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:3e81884229e8dc6b7f258169ec8da246
@SQ     SN:chr17        LN:78774742
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:2a5c95ed99c5298bb107f313c7044588
@SQ     SN:chr18        LN:76117153
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:3d11df432bcdc1407835d5ef2ce62634
```

```
@SQ     SN:chr19        LN:63811651
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:2f1a59077cfad51df907ac25723bff28
@SQ     SN:chr20        LN:62435964
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:f126cdf8a6e0c7f379d618ff66beb2da
@SQ     SN:chr21        LN:46944323
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:f1b74b7f9f4cdbaeb6832ee86cb426c6
@SQ     SN:chr22        LN:49691432
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:2041e6a0c914b48dd537922cca63acb8
@SQ     SN:chrX LN:154913754
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:d7e626c80ad172a4d7c95aadb94d9040
@SQ     SN:chrY LN:57772954
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:62f69d0e82a12af74bad85e2e4a8bd91
@SQ     SN:chr1_random  LN:1663265
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:cc05cb1554258add2eb62e88c0746394
@SQ     SN:chr2_random  LN:185571
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:18ceab9e4667a25c8a1f67869a4356ea
@SQ     SN:chr3_random  LN:749256
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:9cc571e918ac18afa0b2053262cadab6
@SQ     SN:chr4_random  LN:842648
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:9cab2949ccf26ee0f69a875412c93740
@SQ     SN:chr5_random  LN:143687
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:05926bdbff978d4a0906862eb3f773d0
@SQ     SN:chr6_random  LN:1875562
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:d62eb2919ba7b9c1d382c011c5218094
@SQ     SN:chr7_random  LN:549659
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:28ebfb89c858edbc4d71ff3f83d52231
@SQ     SN:chr8_random  LN:943810
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:0ed5b088d843d6f6e6b181465b9e82ed
@SQ     SN:chr9_random  LN:1146434
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:1e3d2d2f141f0550fa28a8d0ed3fd1cf
@SQ     SN:chr10_random LN:113275
```

```
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:50be2d2c6720dabeff497ffb53189daa
@SQ     SN:chr11_random LN:215294
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:bfc93adc30c621d5c83eee3f0d841624
@SQ     SN:chr13_random LN:186858
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:563531689f3dbd691331fd6c5730a88b
@SQ     SN:chr15_random LN:784346
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:bf885e99940d2d439d83eba791804a48
@SQ     SN:chr16_random LN:105485
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:dd06ea813a80b59d9c626b31faf6ae7f
@SQ     SN:chr17_random LN:2617613
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:34d5e2005dffdfaaced1d34f60ed8fc2
@SQ     SN:chr18_random LN:4262
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:f3814841f1939d3ca19072d9e89f3fd7
@SQ     SN:chr19_random LN:301858
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:420ce95da035386cc8c63094288c49e2
@SQ     SN:chr21_random LN:1679693
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:a7252115bfe5bb5525f34d039eecd096
@SQ     SN:chr22_random LN:257318
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:4f2d259b82f7647d3b668063cf18378b
@SQ     SN:chrX_random  LN:1719168
UR:file:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/Homo_sapiens_assembly18.fasta
   M5:f4d71e0758986c15e5455bf3e14e5d6f
```

Creating the fasta index file

We use the faidx command in samtools to prepare the fasta index file. This file describes byte offsets in the fasta file for each contig, allowing us to compute exactly where a particular reference base at contig:pos is in the fasta file.

```
> samtools faidx Homo_sapiens_assembly18.fasta
108.446u 3.384s 2:44.61 67.9%   0+0k 0+0io 0pf+0w
```

This produces a text file with one record per line for each of the fasta contigs. Each record is of the: contig, size, location, basesPerLine, bytesPerLine. The index file produced above looks like:

```
> cat Homo_sapiens_assembly18.fasta.fai
chrM    16571   6       50      51
```

```
chr1    247249719       16915   50      51
chr2    242951149       252211635       50      51
chr3    199501827       500021813       50      51
chr4    191273063       703513683       50      51
chr5    180857866       898612214       50      51
chr6    170899992       1083087244      50      51
chr7    158821424       1257405242      50      51
chr8    146274826       1419403101      50      51
chr9    140273252       1568603430      50      51
chr10   135374737       1711682155      50      51
chr11   134452384       1849764394      50      51
chr12   132349534       1986905833      50      51
chr13   114142980       2121902365      50      51
chr14   106368585       2238328212      50      51
chr15   100338915       2346824176      50      51
chr16   88827254        2449169877      50      51
chr17   78774742        2539773684      50      51
chr18   76117153        2620123928      50      51
chr19   63811651        2697763432      50      51
chr20   62435964        2762851324      50      51
chr21   46944323        2826536015      50      51
chr22   49691432        2874419232      50      51
chrX    154913754       2925104499      50      51
chrY    57772954        3083116535      50      51
chr1_random     1663265 3142044962      50      51
chr2_random     185571  3143741506      50      51
chr3_random     749256  3143930802      50      51
chr4_random     842648  3144695057      50      51
chr5_random     143687  3145554571      50      51
chr6_random     1875562 3145701145      50      51
chr7_random     549659  3147614232      50      51
chr8_random     943810  3148174898      50      51
chr9_random     1146434 3149137598      50      51
chr10_random    113275  3150306975      50      51
chr11_random    215294  3150422530      50      51
chr13_random    186858  3150642144      50      51
chr15_random    784346  3150832754      50      51
chr16_random    105485  3151632801      50      51
chr17_random    2617613 3151740410      50      51
chr18_random    4262    3154410390      50      51
chr19_random    301858  3154414752      50      51
chr21_random    1679693 3154722662      50      51
chr22_random    257318  3156435963      50      51
chrX_random     1719168 3156698441      50      51
```

How can I submit a patch to the GATK codebase? #1267

Last updated on 2012-10-18 15:03:17

The GATK is an open source project that has greatly benefited from the contributions of outside users. The GATK team welcomes contributions from anyone who produces useful functionality in line with the goals of the toolkit. You are welcome to branch the GATK main repository and develop your own tools. Sometimes these tools may be useful to the GATK user community and you may want to make it part of the main GATK distribution. If so we ask you to follow our guidelines for submission of patches.

1. Good practices

There are a few good GIT practices that you should follow to simplify the ultimate goal, which is, adding your changes to the main GATK repository.

- Use branches. Every time you start new work that you are going to submit to the GATK team later, do it in a new branch. Make it a habit as this will simplify many of the following procedures and allow your master branch to always be a fresh (up to date) copy of the GATK main repository. Take a look on [[#How to create a new submission| how to create a new branch for submission]].
- Never merge. Merging creates a branched history with multiple parent nodes that make history hard to understand, impossible to modify and patches near-impossible to create. Merges are very useful when you need to combine multiple repositories and it should ''only'' be used when it makes sense. This means '''never merge''' and '''never pull''' (if it's not a fast-forward, or you will create a merge).
- Commit as often as possible. Every change, should be committed to make sure you can go back in time effectively in your own tree. The commit messages don't matter to us as long as they're meaningful to you in this stage. You can essentially do whatever you want in your local tree with your commits, as long as you don't merge.
- Rebase constantly Your branch is diverging from the master by the minute, so if you keep rebasing as often as you can, you will avoid major conflicts when it's time to send the patches. Take a look at our guide on [[#How to rebase | how to rebase]].
- Tell a meaningful story When it's time to submit your patches to us, reorder your commits and write meaningful commit messages. Each commit must be (as much as possible) self contained. These commits must tell a meaningful story to us so we can understand what it is you're adding to the codebase. Take a look at an [[#How to make your commits | example commit scenario]].
- Generate patches and email them to the group This part is super easy, provided you've followed the good practices. You just have to [[#How to generate the patches | generate the patches]] and e-mail them to gsa-patches@broadinstitute.org.

2. How to create a new submission

You should always start your code by creating a new branch from the most recent version of the main repository with :

```
git checkout master                     (make sure you are in the master branch)
git fetch && git rebase origin/master   (you can substitute this line for "git pull" if
you have no changes in the master branch)
git checkout -b newtool                 (create a new branch for your new tool)
```

Note: If you have submitted a patch to the group, do not continue development on the same branch as we cannot guarantee that your changes will make it to the main repository unchanged.

3. How to rebase

Every time before you rebase, you have to update your copy of the main repository. To do this use:

```
git fetch
```

If you are just trying to keep up with the changes in the main repository after a fetch, you can rebase your branch at anytime using (and this should be all you need to do):

```
git rebase origin/master
```

In case there are conflicts, resolve them as you would and do:

```
git rebase --continue
```

If you don't know how to resolve the conflicts, you can always safely abort the whole process and go back to your branch before you started rebasing:

```
git rebase --abort
```

If you are done and want to generate your patches conforming to the latest repository changes, to edit, squash and reorder your commits use :

```
git rebase -i origin/master
```

At the prompt, you can follow the instructions to squash, edit and reorder accordingly. You can also do this step from IntelliJ with a visual editor that allows you to select what to edit/squash/reorder. You can also take a look at this nice tutorial on how to use interactive rebase.

4. How to make your commits

It is okay to have a list of commits (numbered) somewhat like this in your local tree:

- added function X
- fixed a b and c on X
- b was actually d
- started creating feature Y but had to go to the bathroom
- added Y
- found bug in X, fixed with e
- added Z
- fixed bug in Z with f

Before you can send your tools to us, you have to organize these commits so they tell a meaningful history and are self contained. To achieve this you will need to rebase so you can squash, edit and reorder your commits. This tree makes a lot of sense for your development process, but it makes no sense in the main repository history as it becomes hard to pick/revert commits and understand the history at a glance. After rebasing, you should edit your commits to look like this:

- added X (including commits 2, 3 and 6)
- added Y (including commits 4 and 5)
- added Z (including commits 7 and 8)

Use your commit messages wisely to help quick processing of your patches. Make sure the first line of your commit messages have less than 50 characters (title). Add a blank line and write a paragraph or more explaining what this commit represents (now that it is a package of multiple commits. It is important to have the 50 char title because this is all we see when we look at an extended history to find bugs and it is also our quick access to remember what the commit does to the repository.

A patch should be self contained. Meaning if we decide to adopt feature X and Z but not Y, we should be able to do so by only applying patches 1 and 2. If your patches are co-dependent, you should say so in the commits and justify why you didn't squash the commits together into one tool.

5. How to generate the patches

To generate patches, use :

```
git format-patch since
```

The since parameter is the last commit you want to generate patches from, for example: HEAD^3 will generate patches for HEAD^2, HEAD^1 and HEAD. You can also specify the commit by its id or by using the head of a branch. This is where using branches will make your life easier. If master is always up to date with the main repo with no changes, you can do:

```
git format-patch master   (provided your master is up to date)
```

This will generate a patch for each commit you've created and you can simply e-mail them as an attachment to us.

How can I turn on or customize forum notifications? #27

Last updated on 2012-10-18 15:07:34

By default, the forum does not send notification messages about new comments or discussions. If you want to turn on notifications or cutomize the type of notifications you want, you need to do the following: Go to your profile page by clicking on your user name; Click on Edit profile; In the menu on the left, click on Notification Preferences; Select the categories that you want to follow and the type of notification you want to receive. Be sure to click on Save Preferences.

How can I use parallelism to make GATK tools run faster? #1975

Last updated on 2013-04-19 17:04:34

This document provides technical details and recommendations on how the parallelism options offered by the GATK can be used to yield optimal performance results.

Overview

As explained in the primer on parallelism for the GATK, there are two main kinds of parallelism that can be applied to the GATK: multi-threading and scatter-gather (using Queue).

Multi-threading options

There are two options for multi-threading with the GATK, controlled by the arguments `-nt` and `-nct`, respectively, which can be combined:

- `--nt / --num_threads` controls the number of **data threads** sent to the processor
- `--nct / --num_cpu_threads_per_data_thread` controls the number of **CPU threads** allocated to each data thread

For more information on how these multi-threading options work, please read the primer on parallelism for the GATK.

Memory considerations for multi-threading

Each data thread needs to be given the full amount of memory youâ€™d normally give a single run. So if youâ€™re running a tool that normally requires 2 Gb of memory to run, if you use `-nt 4`, the multithreaded run will use 8 Gb of memory. In contrast, CPU threads will share the memory allocated to their â€œmotherâ€• data thread, so you donâ€™t need to worry about allocating memory based on the number of CPU threads you use.

Additional consideration when using `-nct` with versions 2.2 and 2.3

Because of the way the `-nct` option was originally implemented, in versions 2.2 and 2.3, there is one CPU thread that is reserved by the system to â€œmanageâ€• the rest. So if you use `-nct`, youâ€™ll only really start seeing a speedup with `-nct 3` (which yields two effective "working" threads) and above. This limitation has been resolved in the implementation that will be available in versions 2.4 and up.

Scatter-gather

For more details on scatter-gather, see the primer on parallelism for the GATK and the Queue documentation.

Applicability of parallelism to the major GATK tools

Please note that not all tools support all parallelization modes. The parallelization modes that are available for each tool depend partly on the type of traversal that the tool uses to walk through the data, and partly on the nature of the analyses it performs.

Tool	Full name	Type of traversal	NT	NCT	SG
RTC	RealignerTargetCreator	RodWalker	+	-	-
IR	IndelRealigner	ReadWalker	-	-	+
BR	BaseRecalibrator	LocusWalker	-	+	+
PR	PrintReads	ReadWalker	-	+	-
RR	ReduceReads	ReadWalker	-	-	+
UG	UnifiedGenotyper	LocusWalker	+	+	+

Recommended configurations

The table below summarizes configurations that we typically use for our own projects (one per tool, except we give three alternate possibilities for the UnifiedGenotyper). The different values allocated for each tool reflect not only the technical capabilities of these tools (which options are supported), but also our empirical observations of what provides the best tradeoffs between performance gains and commitment of resources. Please note however that this is meant only as a guide, and that we cannot give you any guarantee that these configurations are the best for your own setup. You will probably have to experiment with the settings to find the configuration that is right for you.

Tool	RTC	IR	BR	PR	RR	UG
Available modes	NT	SG	NCT,SG	NCT	SG	NT,NCT,SG
Cluster nodes	1	4	4	1	4	4 4 4
CPU threads (-nct)	1	1	8	4-8	1	3 6 24
Data threads (-nt)	24	1	1	1	1	8 4 1
Memory (Gb)	48	4	4	4	4	32 16 4

Where NT is data multithreading, NCT is CPU multithreading and SG is scatter-gather using Queue. For more details on scatter-gather, see the primer on parallelism for the GATK and the Queue documentation.

How do I submit a detailed bug report? #1894

Last updated on 2012-12-30 17:17:09

Note: only do this if you have been explicitly asked to do so.

Scenario:

You posted a question about a problem you had with GATK tools, we answered that we think it's a bug, and we asked you to submit a detailed bug report.

Here's what you need to provide:

- The exact command line that you used when you had the problem (in a text file)

- The full stack trace (program output in the console) from the start of the run to the end or error message (in a text file)

- A snippet of the BAM file if applicable and the index (.bai) file associated with it

- If a non-standard reference (i.e. not available in our resource bundle) was used, we need the .fasta, .fai, and .dict files for the reference

- Any other relevant files such as recalibration plots

A snippet file is a slice of the original BAM file which contains the problematic region and is sufficient to reproduce the error. We need it in order to reproduce the problem on our end, which is the first necessary step to finding and fixing the bug. We ask you to provide this as a snippet rather than the full file so that you don't have to upload (and we don't have to process) huge giga-scale files.

Here's how you create a snippet file:

- Look at the error message and see if it cites a specific position where the error occurred

- If not, identify what region caused the problem by running with `-L` argument and progressively narrowing down the interval

- Once you have the region, use PrintReads with `-L` to write the problematic region (with 500 bp padding on either side) to a new file -- this is your snippet file.

- Test your command line on this snippet file to make sure you can still reproduce the error on it.

And finally, here's how you send us the files:

- Put all those files into a `.zip` or `.tar.gz` archive

- Upload them onto our FTP server as explained here (make sure you use the proper UPLOAD credentials)

- Post in the original discussion thread that you have done this

- Be sure to tell us the name of your archive file!

We will get back to you --hopefully with a bug fix!-- as soon as we can.

How does the GATK handle these huge NGS datasets? #1320

Last updated on 2012-10-18 14:57:10

Imagine a simple question like, "What's the depth of coverage at position A of the genome?"

First, you are given billions of reads that are aligned to the genome but not ordered in any particular way (except perhaps in the order they were emitted by the sequencer). This simple question is then very difficult to answer efficiently, because the algorithm is forced to examine every single read in succession, since any one of them might span position A. The algorithm must now take several hours in order to compute this value.

Instead, imagine the billions of reads are now sorted in reference order (that is to say, on each chromosome, the reads are stored on disk in the same order they appear on the chromosome). Now, answering the question above is trivial, as the algorithm can jump to the desired location, examine only the reads that span the position, and return immediately after those reads (and only those reads) are inspected. The total number of reads that need to be interrogated is only a handful, rather than several billion, and the processing time is seconds, not hours.

This reference-ordered sorting enables the GATK to process terabytes of data quickly and without tremendous memory overhead. Most GATK tools run very quickly and with less than 2 gigabytes of RAM. Without this sorting, the GATK cannot operate correctly. Thus, it is a fundamental rule of working with the GATK, which is the reason for the Central Dogma of the GATK:

All datasets (reads, alignments, quality scores, variants, dbSNP information, gene tracks, interval lists - everything) must be sorted in order of one of the canonical references sequences.

How should I interpret VCF files produced by the GATK? #1268

Last updated on 2013-09-05 16:37:49

1. What is VCF?

VCF stands for Variant Call Format. It is a standardized text file format for representing SNP, indel, and structural variation calls. See this page for detailed specifications.

VCF is the primary (and only well-supported) format used by the GATK for variant calls. We prefer it above all others because while it can be a bit verbose, the VCF format is **very explicit** about the exact type and sequence of variation as well as the genotypes of multiple samples for this variation.

That being said, this highly detailed information can be challenging to understand. The information provided by the GATK tools that infer variation from NGS data, such as the UnifiedGenotyper and the HaplotypeCaller, is especially complex. This document describes some specific features and annotations used in the VCF files output by the GATK tools.

2. Basic structure of a VCF file

The following text is a valid VCF file describing the first few SNPs found by the UG in a deep whole genome data set from our favorite test sample, NA12878:

```
##fileformat=VCFv4.0
##FILTER=<ID=LowQual,Description="QUAL < 50.0">
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt
alleles in the order listed">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth (only filtered reads used for
calling)">
##FORMAT=<ID=GQ,Number=1,Type=Float,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PL,Number=3,Type=Float,Description="Normalized, Phred-scaled likelihoods for
AA,AB,BB genotypes where A=ref and B=alt; not applicable if site is not biallelic">
##INFO=<ID=AC,Number=.,Type=Integer,Description="Allele count in genotypes, for each ALT
allele, in the same order as listed">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency, for each ALT allele, in the
same order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called
genotypes">
```

```
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP Membership">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=DS,Number=0,Type=Flag,Description="Were any of the samples downsampled?">
##INFO=<ID=Dels,Number=1,Type=Float,Description="Fraction of Reads Containing Spanning
Deletions">
##INFO=<ID=HRun,Number=1,Type=Integer,Description="Largest Contiguous Homopolymer Run of
Variant Allele In Either Direction">
##INFO=<ID=HaplotypeScore,Number=1,Type=Float,Description="Consistency of the site with two
(and only two) segregating haplotypes">
##INFO=<ID=MQ,Number=1,Type=Float,Description="RMS Mapping Quality">
##INFO=<ID=MQ0,Number=1,Type=Integer,Description="Total Mapping Quality Zero Reads">
##INFO=<ID=QD,Number=1,Type=Float,Description="Variant Confidence/Quality by Depth">
##INFO=<ID=SB,Number=1,Type=Float,Description="Strand Bias">
##INFO=<ID=VQSLOD,Number=1,Type=Float,Description="log10-scaled probability of variant being
true under the trained gaussian mixture model">
##UnifiedGenotyperV2="analysis_type=UnifiedGenotyperV2 input_file=[TEXT CLIPPED FOR
CLARITY]"
#CHROM  POS ID      REF ALT QUAL    FILTER  INFO    FORMAT  NA12878
chr1    873762  .       T   G   5231.78 PASS
AC=1;AF=0.50;AN=2;DP=315;Dels=0.00;HRun=2;HaplotypeScore=15.11;MQ=91.05;MQ0=15;QD=16.61;SB=-
1533.02;VQSLOD=-1.5473 GT:AD:DP:GQ:PL   0/1:173,141:282:99:255,0,255
chr1    877664  rs3828047   A   G   3931.66 PASS
AC=2;AF=1.00;AN=2;DB;DP=105;Dels=0.00;HRun=1;HaplotypeScore=1.59;MQ=92.52;MQ0=4;QD=37.44;SB=
-1152.13;VQSLOD= 0.1185 GT:AD:DP:GQ:PL  1/1:0,105:94:99:255,255,0
chr1    899282  rs28548431  C   T   71.77   PASS
AC=1;AF=0.50;AN=2;DB;DP=4;Dels=0.00;HRun=0;HaplotypeScore=0.00;MQ=99.00;MQ0=0;QD=17.94;SB=-4
6.55;VQSLOD=-1.9148 GT:AD:DP:GQ:PL  0/1:1,3:4:25.92:103,0,26
chr1    974165  rs9442391   T   C   29.84   LowQual
AC=1;AF=0.50;AN=2;DB;DP=18;Dels=0.00;HRun=1;HaplotypeScore=0.16;MQ=95.26;MQ0=0;QD=1.66;SB=-0
.98 GT:AD:DP:GQ:PL  0/1:14,4:14:60.91:61,0,255
```

It seems a bit complex, but the structure of the file is actually quite simple:

```
[HEADER LINES]
#CHROM  POS ID      REF ALT QUAL    FILTER  INFO          FORMAT          NA12878
chr1    873762  .       T   G   5231.78 PASS    [ANNOTATIONS] GT:AD:DP:GQ:PL
0/1:173,141:282:99:255,0,255
chr1    877664  rs3828047   A   G   3931.66 PASS    [ANNOTATIONS] GT:AD:DP:GQ:PL
1/1:0,105:94:99:255,255,0
chr1    899282  rs28548431  C   T   71.77   PASS    [ANNOTATIONS] GT:AD:DP:GQ:PL
0/1:1,3:4:25.92:103,0,26
chr1    974165  rs9442391   T   C   29.84   LowQual [ANNOTATIONS] GT:AD:DP:GQ:PL
0/1:14,4:14:60.91:61,0,255
```

After the header lines and the field names, each line represents a single variant, with various properties of that variant represented in the columns. Note that here everything is a SNP, but some could be indels or CNVs.

3. How variation is represented

The first 6 columns of the VCF, which represent the observed variation, are easy to understand because they have a single, well-defined meaning.

- **CHROM and POS :** The CHROM and POS gives the contig on which the variant occurs. For indels this is actually the base preceding the event, due to how indels are represented in a VCF.

- **ID:** The dbSNP `rs` identifier of the SNP, based on the contig and position of the call and whether a record exists at this site in dbSNP.

- **REF and ALT:** The reference base and alternative base that vary in the samples, or in the population in general. Note that REF and ALT are always given on the forward strand. For indels the REF and ALT bases always include at least one base each (the base before the event).

- **QUAL:** The Phred scaled probability that a REF/ALT polymorphism exists at this site given sequencing data. Because the Phred scale is -10 * log(1-p), a value of 10 indicates a 1 in 10 chance of error, while a 100 indicates a 1 in 10^10 chance. These values can grow very large when a large amount of NGS data is used for variant calling.

- **FILTER:** In a perfect world, the QUAL field would be based on a complete model for all error modes present in the data used to call. Unfortunately, we are still far from this ideal, and we have to use orthogonal approaches to determine which called sites, independent of QUAL, are machine errors and which are real SNPs. Whatever approach is used to filter the SNPs, the VCFs produced by the GATK carry both the PASSing filter records (the ones that are good have PASS in their FILTER field) as well as those that fail (the filter field is anything but PASS or a dot). If the FILTER field is a ".", then no filtering has been applied to the records, meaning that all of the records will be used for analysis but without explicitly saying that any PASS. You should avoid such a situation by always filtering raw variant calls before analysis.

For more details about these fields, please see this page.

In the excerpt shown above, here is how we interpret the line corresponding to each variant:

- chr1:873762 is a novel T/G polymorphism, found with very high confidence (QUAL = 5231.78)

- chr1:877664 is a known A/G SNP (named rs3828047), found with very high confidence (QUAL = 3931.66)

- chr1:899282 is a known C/T SNP (named rs28548431), but has a relative low confidence (QUAL = 71.77)

- chr1:974165 is a known T/C SNP but we have so little evidence for this variant in our data that although we write out a record for it (for book keeping, really) our statistical evidence is so low that we filter the record out as a bad site, as indicated by the "LowQual" annotation.

4. How genotypes are represented

The genotype fields of the VCF look more complicated but they're actually not that hard to interpret once you understand that they're just sets of tags and values. Let's take a look at three of the records shown earlier, simplified to just show the key genotype annotations:

```
chr1    873762  .       T   G   [CLIPPED] GT:AD:DP:GQ:PL    0/1:173,141:282:99:255,0,255
chr1    877664  rs3828047   A   G   [CLIPPED] GT:AD:DP:GQ:PL    1/1:0,105:94:99:255,255,0
chr1    899282  rs28548431  C   T   [CLIPPED] GT:AD:DP:GQ:PL    0/1:1,3:4:25.92:103,0,26
```

Looking at that last column, here is what the tags mean:

- **GT :** The genotype of this sample. For a diploid organism, the GT field indicates the two alleles carried by the sample, encoded by a 0 for the REF allele, 1 for the first ALT allele, 2 for the second ALT allele, etc. When there's a single ALT allele (by far the more common case), GT will be either:

- 0/0 - the sample is homozygous reference

- 0/1 - the sample is heterozygous, carrying 1 copy of each of the REF and ALT alleles

- 1/1 - the sample is homozygous alternate In the three examples above, NA12878 is observed with the allele combinations T/G, G/G, and C/T respectively.

- **GQ:** The Genotype Quality, or Phred-scaled confidence that the true genotype is the one provided in GT. In the diploid case, if GT is 0/1, then GQ is really L(0/1) / (L(0/0) + L(0/1) + L(1/1)), where L is the likelihood that the sample is 0/0, 0/1/, or 1/1 under the model built for the NGS dataset.

- **AD and DP:** These are complementary fields that represent two important ways of thinking about the depth of the data for this sample at this site. See the Technical Documentation for details on AD (DepthPerAlleleBySample) and DP (Coverage).

- **PL:** This field provides the likelihoods of the given genotypes (here, 0/0, 0/1, and 1/1). These are normalized, Phred-scaled likelihoods for each of the 0/0, 0/1, and 1/1, without priors. To be concrete, for the heterozygous case, this is L(data given that the true genotype is 0/1). The most likely genotype (given in the GT field) is scaled so that it's P = 1.0 (0 when Phred-scaled), and the other likelihoods reflect their Phred-scaled likelihoods relative to this most likely genotype.

With that out of the way, let's interpret the genotypes for NA12878 at chr1:899282.

```
chr1    899282  rs28548431  C   T   [CLIPPED] GT:AD:DP:GQ:PL    0/1:1,3:4:25.92:103,0,26
```

At this site, the called genotype is `GT = 0/1`, which is C/T. The confidence indicated by `GQ = 25.92` isn't so good, largely because there were only a total of 4 reads at this site (`DP =4`), 1 of which was REF (=had the reference base) and 3 of which were ALT (=had the alternate base) (indicated by `AD=1,3`). The lack of certainty is evident in the PL field, where `PL(0/1) = 0` (the normalized value that corresponds to a likelihood of 1.0). There's a chance that the subject is "hom-var" (=homozygous with the variant allele) since `PL(1/1) = 26`, which corresponds to 10^(-2.6), or 0.0025, but either way, it's clear that the subject is definitely not "hom-ref" (=homozygous with the reference allele) since `PL(0/0) = 103`, which corresponds to 10^(-10.3), a very small number.

5. Understanding annotations

Finally, variants in a VCF can be annotated with a variety of additional tags, either by the built-in tools or with others that you add yourself. The way they're formatted is similar to what we saw in the Genotype fields, except instead of being in two separate fields (tags and values, respectively) the annotation tags and values are grouped together, so tag-value pairs are written one after another.

```
chr1    873762  [CLIPPED]
AC=1;AF=0.50;AN=2;DP=315;Dels=0.00;HRun=2;HaplotypeScore=15.11;MQ=91.05;MQ0=15;QD=16.61;SB=-
```

```
1533.02;VQSLOD=-1.5473
chr1    877664  [CLIPPED]
AC=2;AF=1.00;AN=2;DB;DP=105;Dels=0.00;HRun=1;HaplotypeScore=1.59;MQ=92.52;MQ0=4;QD=37.44;SB=
-1152.13;VQSLOD= 0.1185
chr1    899282  [CLIPPED]
AC=1;AF=0.50;AN=2;DB;DP=4;Dels=0.00;HRun=0;HaplotypeScore=0.00;MQ=99.00;MQ0=0;QD=17.94;SB=-4
6.55;VQSLOD=-1.9148
```

Here are some commonly used built-in annotations and what they mean:

Annotation tag in VCF	Meaning
AC,AF,AN	See the Technical Documentation for Chromosome Counts.
DB	If present, then the variant is in dbSNP.
DP	See the Technical Documentation for Coverage.
DS	Were any of the samples downsampled because of too much coverage?
Dels	See the Technical Documentation for SpanningDeletions.
MQ and MQ0	See the Technical Documentation for RMS Mapping Quality and Mapping Quality Zero.
BaseQualityRankSumTest	See the Technical Documentation for Base Quality Rank Sum Test.
MappingQualityRankSumTest	See the Technical Documentation for Mapping Quality Rank Sum Test.
ReadPosRankSumTest	See the Technical Documentation for Read Position Rank Sum Test.
HRun	See the Technical Documentation for Homopolymer Run.
HaplotypeScore	See the Technical Documentation for Haplotype Score.
QD	See the Technical Documentation for Qual By Depth.
VQSLOD	Only present when using Variant quality score recalibration. Log odds ratio of being a true variant versus being false under the trained gaussian mixture model.
FS	See the Technical Documentation for Fisher Strand
SB	How much evidence is there for Strand Bias (the variation being seen on only the forward or only the reverse strand) in the reads? Higher SB values denote more bias (and therefore are more likely to indicate false positive calls).

Lane, Library, Sample and Cohort -- what do they mean and why are they important? #3059

Last updated on 2013-08-02 19:12:40

There are four major organizational units for next-generation DNA sequencing processes that used throughout the GATK documentation:

- **Lane:** The basic machine unit for sequencing. The lane reflects the basic independent run of an NGS machine. For Illumina machines, this is the physical sequencing lane.

- **Library:** A unit of DNA preparation that at some point is physically pooled together. Multiple lanes can be run from aliquots from the same library. The DNA library and its preparation is the natural unit that is being sequenced. For example, if the library has limited complexity, then many sequences are duplicated and will

result in a high duplication rate across lanes.

- **Sample:** A single individual, such as human CEPH NA12878. Multiple libraries with different properties can be constructed from the original sample DNA source. Throughout our documentation, we treat samples as independent individuals whose genome sequence we are attempting to determine. Note that from this perspective, tumor / normal samples are different despite coming from the same individual.

- **Cohort:** A collection of samples being analyzed together. This organizational unit is the most subjective and depends very specifically on the design goals of the sequencing project. For population discovery projects like the 1000 Genomes, the analysis cohort is the ~100 individual in each population. For exome projects with many deeply sequenced samples (e.g., ESP with 800 EOMI samples) we divide up the complete set of samples into cohorts of ~50 individuals for multi-sample analyses.

Note that many GATK commands can be run at the lane level, but will give better results seeing all of the data for a single sample, or even all of the data for all samples. Unfortunately, there's a trade-off in computational cost, since running these commands across all of your data simultaneously requires much more computing power. Please see the documentation for each step to understand what is the best way to group or partition your data for that particular process.

Should I use UnifiedGenotyper or HaplotypeCaller to call variants on my data? #3151

Posted on 2013-08-23 21:34:04

The HaplotypeCaller is a more recent and sophisticated tool than the UnifiedGenotyper. Its ability to call SNPs is equivalent to that of the UnifiedGenotyper, and its ability to call indels is far superior. We recommend using HaplotypeCaller in all cases, with only a few exceptions:

- If you want to analyze more than 100 samples at a time (for performance reasons)
- If you are working with non-diploid organisms (UG can handle different levels of ploidy while HC cannot)
- If you are working with pooled samples (also due to the HCâ€™s limitation regarding ploidy)

In those cases, we recommend using UnifiedGenotyper instead of HaplotypeCaller.

What VQSR training sets / arguments should I use for my specific project? #1259

Last updated on 2013-09-11 16:51:03

This document describes the resource datasets and arguments to use in the two steps of VQSR (i.e. the successive application of VariantRecalibrator and ApplyRecalibration), based on our work with human genomes. **Note that VQSR must be run twice in succession in order to build a separate error model for SNPs and INDELs (see the VQSR documentation for more details).**

These recommendations are valid for use with calls generated by both the UnifiedGenotyper and HaplotypeCaller. In the past we made a distinction in how we processed the calls from these two callers, but now we treat them the same way. These recommendations will probably not work properly on calls generated by other (non-GATK) callers.

Resource datasets

The training, truth and known resource datasets mentioned in this document are all available from our resource bundle. If you are working with non-human genomes, you will need to find or generate at least truth and training resource datasets with properties corresponding to those described below.

Resources for SNPs

- True sites training resource: HapMapâ€¨
  This resource is a SNP call set that has been validated to a very high degree of confidence. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). We will also use these sites later on to choose a threshold for filtering variants based on sensitivity to truth sites. The prior likelihood we assign to these variants is Q15 (96.84%).

- True sites training resource: Omniâ€¨
  This resource is a set of polymorphic SNP sites produced by the Omni geno- typing array. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q12 (93.69%).

- Non-true sites training resource: 1000Gâ€¨
  This resource is a set of high-confidence SNP sites produced by the 1000 Genomes Project. The program will consider that the variants in this re- source may contain true variants as well as false positives (truth=false), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q10 (%). 17

- Known sites resource, not used in training: dbSNPâ€¨
  This resource is a call set that has not been validated to a high degree of confidence (truth=false). The program will not use the variants in this resource to train the recalibration model (training=false). However, the program will use these to stratify output metrics such as Ti/Tv ratio by whether variants are present in dbsnp or not (known=true). The prior likelihood we assign to these variants is Q2 (36.90%).

Resources for Indels

- Known and true sites training resource: Mills
  This resource is an Indel call set that has been validated to a high degree of confidence. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q12 (93.69%).

VariantRecalibrator

The variant quality score recalibrator builds an adaptive error model using known variant sites and then applies this model to estimate the probability that each variant is a true genetic variant or a machine artifact. One major improvement from previous recommended protocols is that hand filters do not need to be applied at any point in the process now. All filtering criteria are learned from the data itself.

Common, base command line

```
java -Xmx4g -jar GenomeAnalysisTK.jar \
     -T VariantRecalibrator \
     -R path/to/reference/human_g1k_v37.fasta \
```

```
    -input raw.input.vcf \
    -recalFile path/to/output.recal \
    -tranchesFile path/to/output.tranches \
    -nt 4 \
   [SPECIFY TRUTH AND TRAINING SETS] \
   [SPECIFY WHICH ANNOTATIONS TO USE IN MODELING] \
   [SPECIFY WHICH CLASS OF VARIATION TO MODEL] \
```

SNP specific recommendations

For SNPs we use both HapMap v3.3 and the Omni chip array from the 1000 Genomes Project as training data. In addition we take the highest confidence SNPs from the project's callset. These datasets are available in the GATK resource bundle. Arguments for VariantRecalibrator command:

```
    --numBadVariants 1000 \
    -resource:hapmap,known=false,training=true,truth=true,prior=15.0
hapmap_3.3.b37.sites.vcf \
    -resource:omni,known=false,training=true,truth=true,prior=12.0
1000G_omni2.5.b37.sites.vcf \
    -resource:1000G,known=false,training=true,truth=false,prior=10.0
1000G_phase1.snps.high_confidence.vcf \
    -resource:dbsnp,known=true,training=false,truth=false,prior=2.0 dbsnp.b37.vcf \
    -an QD -an MQRankSum -an ReadPosRankSum -an FS -an DP \
    -mode SNP \
```

Note that, for the above to work, the input vcf needs to be annotated with the corresponding values (QD, FS, DP, etc.). If any of these values are somehow missing, then VariantAnnotator needs to be run first so that VariantRecalibration can run properly.

Also, using the provided sites-only truth data files is important here as parsing the genotypes for VCF files with many samples increases the runtime of the tool significantly.

Important notes about annotations

Some of these annotations might not be the best for your particular dataset. For example, InbreedingCoeff is a population level statistic that requires at least 10 samples in order to be calculated. If your study design has more than 10 samples then it is recommended to be included.

Depth of coverage (the DP annotation invoked by Coverage) should not be used when working with hybrid capture datasets since there is extreme variation in the depth to which targets are captured! In whole genome experiments this variation is indicative of error but that is not the case in capture experiments.

Additionally, the UnifiedGenotyper produces a statistic called the HaplotypeScore which should be used for SNPs. This statistic isn't necessary for the HaplotypeCaller because that mathematics is already built into the likelihood function itself when calling full haplotypes.

Important notes for exome capture experiments

In our testing we've found that in order to achieve the best exome results one needs to use an exome SNP

and/or indel callset with at least 30 samples. For users with experiments containing fewer exome samples there are several options to explore:

- Add additional samples for variant calling, either by sequencing additional samples or using publicly available exome bams from the 1000 Genomes Project (this option is used by the Broad exome production pipeline)
- Use the VQSR with the smaller variant callset but experiment with the precise argument settings (try adding --maxGaussians 4 to your command line, for example, or decrease --numBadVariants)

Indel specific recommendations

When modeling indels with the VQSR we use a training dataset that was created at the Broad by strictly curating the (Mills, Devine, Genome Research, 2011) dataset as as well as adding in very high confidence indels from the 1000 Genomes Project. This dataset is available in the GATK resource bundle. Arguments for VariantRecalibrator:

```
     --maxGaussians 4 --numBadVariants 1000 \
     -resource:mills,known=false,training=true,truth=true,prior=12.0
Mills_and_1000G_gold_standard.indels.b37.sites.vcf \
     -resource:dbsnp,known=true,training=false,truth=false,prior=2.0 dbsnp.b37.vcf \
     -an DP -an FS -an ReadPosRankSum -an MQRankSum \
     -mode INDEL \
```

Note that indels use a different set of annotations than SNPs. Most annotations related to mapping quality have been removed since there is a conflation with the length of an indel in a read and the degradation in mapping quality that is assigned to the read by the aligner. This covariation is not necessarily indicative of being an error in the same way that it is for SNPs.

InbreedingCoeff is a population level statistic that requires at least 10 samples in order to be calculated. If your study design has more than 10 samples then it is recommended to be included.

ApplyRecalibration

The power of the VQSR is that it assigns a calibrated probability to every putative mutation in the callset. The user is then able to decide at what point on the theoretical ROC curve their project wants to live. Some projects, for example, are interested in finding every possible mutation and can tolerate a higher false positive rate. On the other hand, some projects want to generate a ranked list of mutations that they are very certain are real and well supported by the underlying data. The VQSR provides the necessary statistical machinery to effectively apply this sensitivity/specificity tradeoff.

Common, base command line

```
 java -Xmx3g -jar GenomeAnalysisTK.jar \
     -T ApplyRecalibration \
     -R reference/human_g1k_v37.fasta \
     -input raw.input.vcf \
```

```
    -tranchesFile path/to/input.tranches \
    -recalFile path/to/input.recal \
    -o path/to/output.recalibrated.filtered.vcf \
  [SPECIFY THE DESIRED LEVEL OF SENSITIVITY TO TRUTH SITES] \
  [SPECIFY WHICH CLASS OF VARIATION WAS MODELED] \
```

SNP specific recommendations

For SNPs we used HapMap 3.3 and the Omni 2.5M chip as our truth set. We typically seek to achieve 99.9% sensitivity to the accessible truth sites, but this is by no means universally applicable: you will need to experiment to find out what tranche cutoff is right for your data. Generally speaking, projects involving a higher degree of diversity in terms of world populations can expect to achieve a higher truth sensitivity than projects with a smaller scope.

```
    --ts_filter_level 99.9 \
    -mode SNP \
```

Indel specific recommendations

For indels we use the Mills / 1000 Genomes indel truth set described above. We typically seek to achieve 99.9% sensitivity to the accessible truth sites, but this is by no means universally applicable: you will need to experiment to find out what tranche cutoff is right for your data. Generally speaking, projects involving a higher degree of diversity in terms of world populations can expect to achieve a higher truth sensitivity than projects with a smaller scope.

```
    --ts_filter_level 99.9 \
    -mode INDEL \
```

What are JEXL expressions and how can I use them with the GATK? #1255

Last updated on 2013-07-25 20:56:10

1. JEXL in a nutshell

JEXL stands for Java EXpression Language. It's not a part of the GATK as such; it's a software library that can be used by Java-based programs like the GATK. It can be used for many things, but in the context of the GATK, it has one very specific use: making it possible to operate on subsets of variants from VCF files based on one or more annotations, using a single command. This is typically done with walkers such as VariantFiltration and SelectVariants.

2. Basic structure of JEXL expressions for use with the GATK

In this context, a JEXL expression is a string (in the computing sense, i.e. a series of characters) that tells the GATK which annotations to look at and what selection rules to apply.

JEXL expressions contain three basic components: keys and values, connected by operators. For example, in this simple JEXL expression which selects variants whose quality score is greater than 30:

```
"QUAL > 30.0"
```

- QUAL is a **key**: the name of the annotation we want to look at
- 30.0 is a **value**: the threshold that we want to use to evaluate variant quality against
- > is an **operator**: it determines which "side" of the threshold we want to select

The complete expression must be framed by double quotes. Within this, keys are strings (typically written in uppercase or CamelCase), and values can be either strings, numbers or booleans (TRUE or FALSE) -- but if they are strings the values must be framed by single quotes, as in the following example:

```
"MY_STRING_KEY == 'foo'"
```

3. Evaluation on multiple annotations

You can build expressions that calculate a metric based on two separate annotations, for example if you want to select variants for which quality (QUAL) divided by depth of coverage (DP) is below a certain threshold value:

```
"QUAL / DP < 10.0"
```

You can also join multiple conditional statements with logical operators, for example if you want to select variants that have both sufficient quality (QUAL) **and** a certain depth of coverage (DP):

```
"QUAL > 30.0 && DP == 10"
```

where && is the logical "AND".

Or if you want to select variants that have at least one of several conditions fulfilled:

```
"QD < 2.0 || ReadPosRankSum < -20.0 || FS > 200.0"
```

where || is the logical "OR".

4. Important caveats

Sensitivity to case and type

- Case

Currently, VCF INFO field keys are case-sensitive. That means that if you have a QUAL field in uppercase in your VCF record, the system will not recognize it if you write it differently (Qual, qual or whatever) in your JEXL expression.

- Type

The types (i.e. string, integer, non-integer or boolean) used in your expression must be exactly the same as that

of the value you are trying to evaluate. In other words, if you have a QUAL field with non-integer values (e.g. **45.3**) and your filter expression is written as an integer (e.g. "QUAL < **50**"), the system will throw a hissy fit (aka a Java exception).

Complex queries

We highly recommend that complex expressions involving multiple AND/OR operations be split up into separate expressions whenever possible to avoid confusion. If you are using complex expressions, make sure to test them on a panel of different sites with several combinations of yes/no criteria.

5. More complex JEXL magic

Note that this last part is fairly advanced and not for the faint of heart. To be frank, it's also explained rather more briefly than the topic deserves. But if there's enough demand for this level of usage (click the "view in forum" link and leave a comment) we'll consider producing a full-length tutorial.

Accessing the underlying VariantContext directly

If you are familiar with the VariantContext, Genotype and its associated classes and methods, you can directly access the full range of capabilities of the underlying objects from the command line. The underlying VariantContext object is available through the `vc` variable.

For example, suppose I want to use SelectVariants to select all of the sites where sample NA12878 is homozygous-reference. This can be accomplished by assessing the underlying VariantContext as follows:

```
java -Xmx4g -jar GenomeAnalysisTK.jar -T SelectVariants -R b37/human_g1k_v37.fasta --variant my.vcf -select 'vc.getGenotype("NA12878").isHomRef()'
```

Groovy, right? Now here's a more sophisticated example of JEXL expression that finds all novel variants in the total set with allele frequency > 0.25 but not 1, is not filtered, and is non-reference in 01-0263 sample:

```
! vc.getGenotype("01-0263").isHomRef() && (vc.getID() == null || vc.getID().equals(".")) && AF > 0.25 && AF < 1.0 && vc.isNotFiltered() && vc.isSNP() -o 01-0263.high_freq_novels.vcf -sn 01-0263
```

Using the VariantContext to evaluate boolean values

The classic way of evaluating a boolean goes like this:

```
java -Xmx4g -jar GenomeAnalysisTK.jar -T SelectVariants -R b37/human_g1k_v37.fasta --variant my.vcf -select 'DB'
```

But you can also use the VariantContext object like this:

```
java -Xmx4g -jar GenomeAnalysisTK.jar -T SelectVariants -R b37/human_g1k_v37.fasta --variant my.vcf -select 'vc.hasAttribute("DB")'
```

6. Using JEXL to evaluate arrays

Sometimes you might want to write a JEXL expression to evaluate e.g. the AD (allelic depth) field in the FORMAT column. However, the AD is technically not an integer; rather it is a list (array) of integers. One can evaluate the array data using the "." operator. Here's an example:

```
java -Xmx4g -jar GenomeAnalysisTK.jar -T SelectVariants -R b37/human_g1k_v37.fasta --variant my.vcf -select 'vc.getGenotype("NA12878").getAD().0 > 10'
```

What are the prerequisites for running GATK? #1852

Last updated on 2013-10-03 15:57:34

1. Operating system

The GATK runs natively on most if not all flavors of UNIX, which includes MacOSX, Linux and BSD. It is possible to get it running on Windows using Cygwin, but we don't provide any support nor instructions for that.

2. Java

The GATK is a Java-based program, so you'll need to have Java installed on your machine. The Java version should be at 1.6 (at this time we don't support 1.7). You can check what version you have by typing `java -version` at the command line. This article has some more details about what to do if you don't have the right version. Note that at this time we only support the Sun/Oracle Java JDK; OpenJDK is not supported.

3. Familiarity with command-line programs

The GATK does not have a Graphical User Interface (GUI). You don't open it by clicking on the `.jar` file; you have to use the Console (or Terminal) to input commands. If this is all new to you, we recommend you first learn about that and follow some online tutorials before trying to use the GATK. It's not difficult but you'll need to learn some jargon and get used to living without a mouse...

4. Other

Some of the GATK tools produce plots using R, so if you want to get the plots you'll need to have R installed, as well as several R libraries. Full details can be found in the Tutorial on prerequisites.

What do the VariantEval modules do? #2361

Last updated on 2013-03-18 20:26:03

VariantEval accepts two types of modules: stratification and evaluation modules.

- Stratification modules will stratify (group) the variants based on certain properties.
- Evaluation modules will compute certain metrics for the variants

CpG

CpG is a three-state stratification:

- The locus is a CpG site ("CpG")
- The locus is not a CpG site ("non_CpG")
- The locus is either a CpG or not a CpG site ("all")

A CpG site is defined as a site where the reference base at a locus is a C and the adjacent reference base in the 3' direction is a G.

EvalRod

EvalRod is an N-state stratification, where N is the number of eval rods bound to VariantEval.

Sample

Sample is an N-state stratification, where N is the number of samples in the eval files.

Filter

Filter is a three-state stratification:

- The locus passes QC filters ("called")
- The locus fails QC filters ("filtered")
- The locus either passes or fails QC filters ("raw")

FunctionalClass

FunctionalClass is a four-state stratification:

- The locus is a synonymous site ("silent")
- The locus is a missense site ("missense")
- The locus is a nonsense site ("nonsense")
- The locus is of any functional class ("any")

CompRod

CompRod is an N-state stratification, where N is the number of comp tracks bound to VariantEval.

Degeneracy

Degeneracy is a six-state stratification:

- The underlying base position in the codon is 1-fold degenerate ("1-fold")
- The underlying base position in the codon is 2-fold degenerate ("2-fold")
- The underlying base position in the codon is 3-fold degenerate ("3-fold")

- The underlying base position in the codon is 4-fold degenerate ("4-fold")
- The underlying base position in the codon is 6-fold degenerate ("6-fold")
- The underlying base position in the codon is degenerate at any level ("all")

See the [http://en.wikipedia.org/wiki/Genetic_code#Degeneracy Wikipedia page on degeneracy] for more information.

JexlExpression

JexlExpression is an N-state stratification, where N is the number of JEXL expressions supplied to VariantEval. See [[Using JEXL expressions]]

Novelty

Novelty is a three-state stratification:

- The locus overlaps the knowns comp track (usually the dbSNP track) ("known")
- The locus does not overlap the knowns comp track ("novel")
- The locus either overlaps or does not overlap the knowns comp track ("all")

CountVariants

CountVariants is an evaluation module that computes the following metrics:

Metric	Definition
nProcessedLoci	Number of processed loci
nCalledLoci	Number of called loci
nRefLoci	Number of reference loci
nVariantLoci	Number of variant loci
variantRate	Variants per loci rate
variantRatePerBp	Number of variants per base
nSNPs	Number of snp loci
nInsertions	Number of insertion
nDeletions	Number of deletions
nComplex	Number of complex loci
nNoCalls	Number of no calls loci
nHets	Number of het loci
nHomRef	Number of hom ref loci
nHomVar	Number of hom var loci
nSingletons	Number of singletons
heterozygosity	heterozygosity per locus rate
heterozygosityPerBp	heterozygosity per base pair
hetHomRatio	heterozygosity to homozygosity ratio
indelRate	indel rate (insertion count + deletion count)
indelRatePerBp	indel rate per base pair
deletionInsertionRatio	deletion to insertion ratio

CompOverlap

CompOverlap is an evaluation module that computes the following metrics:

Metric	Definition
nEvalSNPs	number of eval SNP sites
nCompSNPs	number of comp SNP sites
novelSites	number of eval sites outside of comp sites
nVariantsAtComp	number of eval sites at comp sites (that is, sharing the same locus as a variant in the comp track, regardless of whether the alternate allele is the same)
compRate	percentage of eval sites at comp sites
nConcordant	number of concordant sites (that is, for the sites that share the same locus as a variant in the comp track, those that have the same alternate allele)
concordantRate	the concordance rate

Understanding the output of CompOverlap

A SNP in the detection set is said to be 'concordant' if the position exactly matches an entry in dbSNP and the allele is the same. To understand this and other output of CompOverlap, we shall examine a detailed example. First, consider a fake dbSNP file (headers are suppressed so that one can see the important things):

```
 $ grep -v '##' dbsnp.vcf
 #CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
 1       10327   rs112750067     T       C       .       .
ASP;R5;VC=SNP;VP=050000020005000000000100;WGT=1;dbSNPBuildID=132
```

Now, a detection set file with a single sample, where the variant allele is the same as listed in dbSNP:

```
 $ grep -v '##' eval_correct_allele.vcf
 #CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT            001-6
 1       10327   .       T       C       5168.52 PASS    ...     GT:AD:DP:GQ:PL
0/1:357,238:373:99:3959,0,4059
```

Finally, a detection set file with a single sample, but the alternate allele differs from that in dbSNP:

```
 $ grep -v '##' eval_incorrect_allele.vcf
 #CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT            001-6
 1       10327   .       T       A       5168.52 PASS    ...     GT:AD:DP:GQ:PL
0/1:357,238:373:99:3959,0,4059
```

Running VariantEval with just the CompOverlap module:

```
 $ java -jar $STING_DIR/dist/GenomeAnalysisTK.jar -T VariantEval \
     -R /seq/references/Homo_sapiens_assembly19/v1/Homo_sapiens_assembly19.fasta \
     -L 1:10327 \
```

```
    -B:dbsnp,VCF dbsnp.vcf \
    -B:eval_correct_allele,VCF eval_correct_allele.vcf \
    -B:eval_incorrect_allele,VCF eval_incorrect_allele.vcf \
    -noEV \
    -EV CompOverlap \
    -o eval.table
```

We find that the eval.table file contains the following:

```
 $ grep -v '##' eval.table | column -t
 CompOverlap  CompRod  EvalRod                JexlExpression  Novelty  nEvalVariants
nCompVariants  novelSites  nVariantsAtComp  compRate      nConcordant  concordantRate
 CompOverlap  dbsnp    eval_correct_allele    none            all      1              1
        0           1                100.00000000  1            100.00000000
 CompOverlap  dbsnp    eval_correct_allele    none            known    1              1
        0           1                100.00000000  1            100.00000000
 CompOverlap  dbsnp    eval_correct_allele    none            novel    0              0
        0           0                0.00000000    0            0.00000000
 CompOverlap  dbsnp    eval_incorrect_allele  none            all      1              1
        0           1                100.00000000  0            0.00000000
 CompOverlap  dbsnp    eval_incorrect_allele  none            known    1              1
        0           1                100.00000000  0            0.00000000
 CompOverlap  dbsnp    eval_incorrect_allele  none            novel    0              0
        0           0                0.00000000    0            0.00000000
```

As you can see, the detection set variant was listed under nVariantsAtComp (meaning the variant was seen at a position listed in dbSNP), but only the eval_correct_allele dataset is shown to be concordant at that site, because the allele listed in this dataset and dbSNP match.

TiTvVariantEvaluator

TiTvVariantEvaluator is an evaluation module that computes the following metrics:

Metric	Definition
nTi	number of transition loci
nTv	number of transversion loci
tiTvRatio	the transition to transversion ratio
nTiInComp	number of comp transition sites
nTvInComp	number of comp transversion sites
TiTvRatioStandard	the transition to transversion ratio for comp sites

What input files does the GATK accept? #1204

Last updated on 2013-09-13 17:36:42

All analyses done with the GATK typically involve several (though not necessarily all) of the following inputs:

- Reference genome sequence
- Sequencing reads
- Intervals of interest
- Reference-ordered data

This article describes the corresponding file formats that are acceptable for use with the GATK.

1. Reference Genome Sequence

The GATK requires the reference sequence in a single reference sequence in FASTA format, with all contigs in the same file. The GATK requires strict adherence to the FASTA standard. All the standard IUPAC bases are accepted, but keep in mind that non-standard bases (i.e. other than ACGT, such as W for example) will be ignored (i.e. those positions in the genome will be skipped).

Some users have reported having issues with reference files that have been stored or modified on Windows filesystems. The issues manifest as "10" characters (corresponding to encoded newlines) inserted in the sequence, which cause the GATK to quit with an error. If you encounter this issue, you will need to re-download a valid master copy of the reference file, or clean it up yourself.

Gzipped fasta files will not work with the GATK, so please make sure to unzip them first. Please see this article for more information on preparing FASTA reference sequences for use with the GATK.

Important note about human genome reference versions

If you are using human data, your reads must be aligned to one of the official b3x (e.g. b36, b37) or hg1x (e.g. hg18, hg19) references. The contig ordering in the reference you used must exactly match that of one of the official references canonical orderings. These are defined by historical karotyping of largest to smallest chromosomes, followed by the X, Y, and MT for the b3x references; the order is thus 1, 2, 3, ..., 10, 11, 12, ... 20, 21, 22, X, Y, MT. The hg1x references differ in that the chromosome names are prefixed with "chr" and chrM appears first instead of last. The GATK will detect misordered contigs (for example, lexicographically sorted) and throw an error. This draconian approach, though unnecessary technically, ensures that all supplementary data provided with the GATK works correctly. You can use ReorderSam to fix a BAM file aligned to a missorted reference sequence.

Our Best Practice recommendation is that you use a standard GATK reference from the GATK resource bundle.

2. Sequencing Reads

The only input format for sequence reads that the GATK itself supports is the [Sequence Alignment/Map (SAM)] format. See [SAM/BAM] for more details on the SAM/BAM format as well as Samtools and Picard, two complementary sets of utilities for working with SAM/BAM files.

If you don't find the information you need in this section, please see our FAQs on BAM files.

If you are starting out your pipeline with raw reads (typically in FASTQ format) you'll need to make sure that when you map those reads to the reference and produce a BAM file, the resulting BAM file is fully compliant with the GATK requirements. See the Best Practices documentation for detailed instructions on how to do this.

In addition to being in SAM format, we require the following additional constraints in order to use your file with the GATK:

- The file must be binary (with `.bam` file extension).
- The file must be indexed.
- The file must be sorted in coordinate order with respect to the reference (i.e. the contig ordering in your bam must exactly match that of the reference you are using).
- The file must have a proper bam header with read groups. Each read group must contain the platform (PL) and sample (SM) tags. For the platform value, we currently support 454, LS454, Illumina, Solid, ABI_Solid, and CG (all case-insensitive).
- Each read in the file must be associated with exactly one read group.

Below is an example well-formed SAM field header and fields (with @SQ dictionary truncated to show only the first two chromosomes for brevity):

```
@HD     VN:1.0  GO:none SO:coordinate
@SQ     SN:1    LN:249250621    AS:NCBI37
UR:file:/lustre/scratch102/projects/g1k/ref/main_project/human_g1k_v37.fasta
M5:1b22b98cdeb4a9304cb5d48026a85128
@SQ     SN:2    LN:243199373    AS:NCBI37
UR:file:/lustre/scratch102/projects/g1k/ref/main_project/human_g1k_v37.fasta
M5:a0d9851da00400dec1098a9255ac712e
@RG     ID:ERR000162    PL:ILLUMINA     LB:g1k-sc-NA12776-CEU-1 PI:200  DS:SRP000031
SM:NA12776      CN:SC
@RG     ID:ERR000252    PL:ILLUMINA     LB:g1k-sc-NA12776-CEU-1 PI:200  DS:SRP000031
SM:NA12776      CN:SC
@RG     ID:ERR001684    PL:ILLUMINA     LB:g1k-sc-NA12776-CEU-1 PI:200  DS:SRP000031
SM:NA12776      CN:SC
@RG     ID:ERR001685    PL:ILLUMINA     LB:g1k-sc-NA12776-CEU-1 PI:200  DS:SRP000031
SM:NA12776      CN:SC
@PG     ID:GATK TableRecalibration      VN:v2.2.16      CL:Covariates=[ReadGroupCovariate,
QualityScoreCovariate, DinucCovariate, CycleCovariate], use_original_quals=true, defau
t_read_group=DefaultReadGroup, default_platform=Illumina, force_read_group=null,
force_platform=null, solid_recal_mode=SET_Q_ZERO, window_size_nqs=5, homopolymer_nback=7,
except on_if_no_tile=false, pQ=5, maxQ=40, smoothing=137
UR:file:/lustre/scratch102/projects/g1k/ref/main_project/human_g1k_v37.fasta
M5:b4eb71ee878d3706246b7c1dbef69299
@PG     ID:bwa  VN:0.5.5
ERR001685.4315085       16      1       9997    25      35M     *       0       0
```

```
CCGATCTCCCTAACCCTAACCCTAACCCTAACCCT     ?8:C7ACAABBCBAAB?CCAABBEBA@ACEBBB@?     XT:A:U
XN:i:4    X0:i:1  X1:i:0  XM:i:2  XO:i:0  XG:i:0  RG:Z:ERR001685  NM:i:6
MD:Z:0N0N0N0N1A0A28     OQ:Z:>>:>2>>>>>>>>>>>>>>>>>>?>>>>??>???>
ERR001689.1165834       117     1       9997    0       *       =       9997    0
CCGATCTAGGGTTAGGGTTAGGGTTAGGGTTAGGG     >7AA<@@C?@?B?B??>9?B??>A?B???BAB??@
RG:Z:ERR001689    OQ:Z:>:<<8<<<><<><><<>7<>>>?>>??>???????
ERR001689.1165834       185     1       9997    25      35M     =       9997    0
CCGATCTCCCTAACCCTAACCCTAACCCTAACCCT     758A:?>>8?=@@>>?;4<>=??@@==??@?==?8     XT:A:U
XN:i:4    SM:i:25 AM:i:0  X0:i:1  X1:i:0  XM:i:2  XO:i:0  XG:i:0  RG:Z:ERR001689  NM:i:6
MD:Z:0N0N0N0N1A0A28     OQ:Z:;74>7><><><>>>>>><:<>>>>>>>>>>>>>>>>
ERR001688.2681347       117     1       9998    0       *       =       9998    0
CGATCTTAGGGTTAGGGTTAGGGTTAGGGTTAGGG     5@BA@A6B???A?B??>B@B??>B@B??>BAB???
RG:Z:ERR001688    OQ:Z:=>>>>><4><<?><?????????????????????
```

Note about fixing BAM files with alternative sortings

The GATK requires that the BAM file be sorted in the same order as the reference. Unfortunately, many BAM files have headers that are sorted in some other order -- lexicographical order is a common alternative. To resort the BAM file please use ReorderSam.

3. Intervals of interest

If you don't find the information you need in this section, please see our FAQs on interval lists.

The GATK accept interval files for processing subsets of the genome in Picard-style interval lists. These files typically have an extension such as `'.list' or more explicitly,`.interval_list`, and look like this:

```
@HD     VN:1.0  SO:coordinate
@SQ     SN:1    LN:249250621    AS:GRCh37
UR:http://www.broadinstitute.org/ftp/pub/seq/references/Homo_sapiens_assembly19.fasta
M5:1b22b98cdeb4a9304cb5d48026a85128     SP:Homo Sapiens
@SQ     SN:2    LN:243199373    AS:GRCh37
UR:http://www.broadinstitute.org/ftp/pub/seq/references/Homo_sapiens_assembly19.fasta
M5:a0d9851da00400dec1098a9255ac712e     SP:Homo Sapiens
1       30366   30503   +       target_1
1       69089   70010   +       target_2
1       367657  368599  +       target_3
1       621094  622036  +       target_4
1       861320  861395  +       target_5
1       865533  865718  +       target_6
...
```

consisting of aSAM-file-like sequence dictionary (the header), and targets in the form of + . These interval lists are tab-delimited. They are also 1-based (first position in the genome is position 1, not position 0). The easiest way to create such a file is to combine your reference file's sequence dictionary (the file stored alongside the reference fasta file with the `.dict` extension) and your intervals into one file.

You can also specify a list of intervals formatted as :- (one interval per line). No sequence dictionary is necessary. This file format also uses 1-based coordinates.

Finally, we also accept BED style interval lists. Warning: this file format is 0-based for the start coordinates, so coordinates taken from 1-based formats should be offset by 1.

4. Reference Ordered Data (ROD) file formats

The GATK can associate arbitrary reference ordered data (ROD) files with named tracks for all tools. Some tools require specific ROD data files for processing, and developers are free to write tools that access arbitrary data sets using the ROD interface. The general ROD system has the following syntax:

```
-argumentName:name,type file
```

Where `name` is the name in the GATK tool (like "eval" in VariantEval), `type` is the type of the file, such as VCF or dbSNP, and `file` is the path to the file containing the ROD data.

The GATK supports several common file formats for reading ROD data:

- VCF : VCF type, the recommended format for representing variant loci and genotype calls. The GATK will only process valid VCF files; VCFTools provides the official VCF validator. See here for a useful poster detailing the VCF specification.
- UCSC formated dbSNP : dbSNP type, UCSC dbSNP database output
- BED : BED type, a general purpose format for representing genomic interval data, useful for masks and other interval outputs. **Please note that the bed format is 0-based while most other formats are 1-based.**

Note that we no longer support the PED format. See here for converting .ped files to VCF.

If you need additional information on VCF files, please see our FAQs on VCF files here and here.

What is "Phone Home" and how does it affect me? #1250

Last updated on 2013-03-25 18:16:42

1. What it is and how it helps us improve the GATK

Since September, 2010, the GATK has had a "phone-home" feature that sends us information about each GATK run via the Broad filesystem (within the Broad) and Amazon's S3 cloud storage service (outside the Broad). This feature is enabled by default.

The information provided by the phone-home feature is critical in driving improvements to the GATK

- By recording detailed information about each error that occurs, it enables GATK developers to **identify and fix previously-unknown bugs** in the GATK. We are constantly monitoring the errors our users encounter and do our best to fix those errors that are caused by bugs in our code.

- It allows us to better understand how the GATK is used in practice and **adjust our documentation and development goals** for common use cases.

- It gives us a picture of **which versions** of the GATK are in use over time, and how successful we've been at encouraging users to migrate from obsolete or broken versions of the GATK to newer, improved versions.

- It tells us **which tools** are most commonly used, allowing us to monitor the adoption of newly-released tools and abandonment of outdated tools.

- It provides us with a sense of the **overall size of our user base** and the major organizations/institutions using the GATK.

2. What information is sent to us

Below are two example GATK Run Reports showing exactly what information is sent to us each time the GATK phones home.

A successful run:

```
<GATK-run-report>
    <id>D7D31ULwTSxlAwnEOSmW6Z4PawXwMxEz</id>
    <start-time>2012/03/10 20.21.19</start-time>
    <end-time>2012/03/10 20.21.19</end-time>
    <run-time>0</run-time>
    <walker-name>CountReads</walker-name>
    <svn-version>1.4-483-g63ecdb2</svn-version>
    <total-memory>85000192</total-memory>
    <max-memory>129957888</max-memory>
    <user-name>depristo</user-name>
    <host-name>10.0.1.10</host-name>
    <java>Apple Inc.-1.6.0_26</java>
    <machine>Mac OS X-x86_64</machine>
    <iterations>105</iterations>
</GATK-run-report>
```

A run where an exception has occurred:

```
<GATK-run-report>
   <id>yX3AnltsqIlXH9kAQqTWHQUd8CQ5bikz</id>
   <exception>
      <message>Failed to parse Genome Location string: 20:10,000,000-10,000,001x</message>
      <stacktrace class="java.util.ArrayList">
         <string>
org.broadinstitute.sting.utils.GenomeLocParser.parseGenomeLoc(GenomeLocParser.java:377)<
/string>
         <string>
org.broadinstitute.sting.utils.interval.IntervalUtils.parseIntervalArguments(IntervalUtils.j
ava:82)</string>
         <string>
org.broadinstitute.sting.commandline.IntervalBinding.getIntervals(IntervalBinding.java:106)<
```

```
/string>
         <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.loadIntervals(GenomeAnalysisEngine.java:6
18)</string>
         <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.initializeIntervals(GenomeAnalysisEngine.
java:585)</string>
         <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.execute(GenomeAnalysisEngine.java:231)<
/string>
         <string>
org.broadinstitute.sting.gatk.CommandLineExecutable.execute(CommandLineExecutable.java:128)<
/string>
         <string>
org.broadinstitute.sting.commandline.CommandLineProgram.start(CommandLineProgram.java:236)<
/string>
         <string>
org.broadinstitute.sting.commandline.CommandLineProgram.start(CommandLineProgram.java:146)<
/string>
         <string>org.broadinstitute.sting.gatk.CommandLineGATK.main(CommandLineGATK.java:92)
</string>
      </stacktrace>
      <cause>
         <message>Position: &apos;10,000,001x&apos; contains invalid chars.</message>
         <stacktrace class="java.util.ArrayList">
            <string>
org.broadinstitute.sting.utils.GenomeLocParser.parsePosition(GenomeLocParser.java:411)<
/string>
            <string>
org.broadinstitute.sting.utils.GenomeLocParser.parseGenomeLoc(GenomeLocParser.java:374)<
/string>
            <string>
org.broadinstitute.sting.utils.interval.IntervalUtils.parseIntervalArguments(IntervalUtils.j
ava:82)</string>
            <string>
org.broadinstitute.sting.commandline.IntervalBinding.getIntervals(IntervalBinding.java:106)<
/string>
            <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.loadIntervals(GenomeAnalysisEngine.java:6
18)</string>
            <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.initializeIntervals(GenomeAnalysisEngine.
java:585)</string>
            <string>
org.broadinstitute.sting.gatk.GenomeAnalysisEngine.execute(GenomeAnalysisEngine.java:231)<
/string>
```

```
            <string>
org.broadinstitute.sting.gatk.CommandLineExecutable.execute(CommandLineExecutable.java:128)<
/string>
            <string>
org.broadinstitute.sting.commandline.CommandLineProgram.start(CommandLineProgram.java:236)<
/string>
            <string>
org.broadinstitute.sting.commandline.CommandLineProgram.start(CommandLineProgram.java:146)<
/string>
            <string>
org.broadinstitute.sting.gatk.CommandLineGATK.main(CommandLineGATK.java:92)</string>
         </stacktrace>
         <is-user-exception>false</is-user-exception>
      </cause>
      <is-user-exception>true</is-user-exception>
   </exception>
   <start-time>2012/03/10 20.19.52</start-time>
   <end-time>2012/03/10 20.19.52</end-time>
   <run-time>0</run-time>
   <walker-name>CountReads</walker-name>
   <svn-version>1.4-483-g63ecdb2</svn-version>
   <total-memory>85000192</total-memory>
   <max-memory>129957888</max-memory>
   <user-name>depristo</user-name>
   <host-name>10.0.1.10</host-name>
   <java>Apple Inc.-1.6.0_26</java>
   <machine>Mac OS X-x86_64</machine>
   <iterations>0</iterations>
</GATK-run-report>
```

Note that as of GATK 1.5 we no longer collect information about the command-line executed, the working directory, or tmp directory.

3. Disabling Phone Home

The GATK is currently in the process of evolving to require interaction with Amazon S3 as a normal part of each run. For this reason, and because the information contained in the GATK run reports is **so critical in driving improvements** to the GATK, we strongly discourage our users from disabling the phone-home feature.

At the same time, we recognize that some of our users do have legitimate reasons for needing to run the GATK with phone-home disabled, and we don't wish to make it impossible for these users to run the GATK.

Examples of legitimate reasons for disabling Phone Home

- Technical reasons: Your local network might have restrictions in place that don't allow the GATK to access external resources, or you might need to run the GATK in a network-less environment.

- Organizational reasons: Your organization's policies might forbid the dissemination of one or more pieces of information contained in the GATK run report.

For such users we have provided an `-et NO_ET` option in the GATK to disable the phone-home feature. To use this option in GATK 1.5 and later, you need to contact us to request a key. Instructions for doing so are below.

How to obtain and use a GATK key

To obtain a GATK key, please fill out the request form.

Running the GATK with a key is simple: you just need to append a `-K your.key` argument to your customary command line, where `your.key` is the path to the key file you obtained from us:

```
java -jar dist/GenomeAnalysisTK.jar \
     -T PrintReads \
     -I public/testdata/exampleBAM.bam \
     -R public/testdata/exampleFASTA.fasta \
     -et NO_ET \
     -K your.key
```

The `-K` argument is only necessary when running the GATK with the `NO_ET` option.

Troubleshooting key-related problems

- Corrupt/Unreadable/Revoked Keys

If you get an error message from the GATK saying that your key is corrupt, unreadable, or has been revoked, please email '"gsahelp@broadinstitute.org"' to ask for a replacement key.

- GATK Public Key Not Found

If you get an error message stating that the GATK public key could not be located or read, then something is likely wrong with your build of the GATK. If you're running the binary release, try downloading it again. If you're compiling from source, try doing an `ant clean` and re-compiling. If all else fails, please ask for help on our community forum.

What does GSA use Phone Home data for?

We use the phone home data for three main purposes. First, we monitor the input logs for errors that occur in the GATK, and proactively fix them in the codebase. Second, we monitor the usage rates of the GATK in general and specific versions of the GATK to explain how widely used the GATK is to funding agencies and other potential supporters. Finally, we monitor adoption rates of specific GATK tools to understand how quickly new tools reach our users. Many of these analyses require us to aggregate the data by unique user, which is why we still collect the username of the individual who ran the GATK (as you can see in the plots). Examples of all three uses are shown in the Tableau graphs below, which update each night and are sent to the GATK members each morning for review.

What is GATK-Lite and how does it relate to "full" GATK 2.x? [RETIRED] #1720

Last updated on 2013-09-13 18:04:45

Please note that GATK-Lite was retired in February 2013 when version 2.4 was released. See the announcement here.

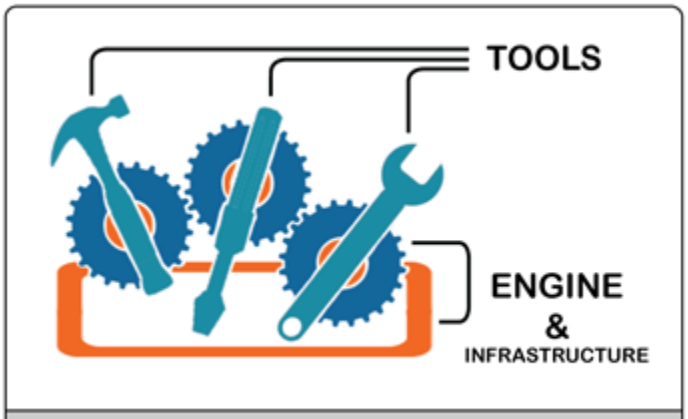
You probably know by now that GATK-Lite is a free-for-everyone and completely open-source version of the GATK (licensed under the original MIT license).

But what's in the box? What can GATK-Lite do -- or rather, what can it **not** do that the full version (let's call it GATK-Full) can? And what does that mean exactly, in terms of functionality, reliability and power?

To really understand the differences between GATK-Lite and GATK-Full, you need some more information on how the GATK works, and how we work to develop and improve it.

First you need to understand what are the two core components of the GATK: the engine and tools (see picture below).

As explained here, the **engine** handles all the common work that's related to data access, conversion and traversal, as well as high-performance computing features. The engine is supported by an infrastructure of software libraries. If the GATK was a car, that would be the engine and chassis. What we call the **tools* are attached on top of that, and they provide the various analytical and processing functionalities like variant calling and base or variant recalibration. On your car, that would be headlights, airbags and so on.



The core GATK components

Second is how we work on developing the GATK, and what it means for how improvements are shared (or not) between Lite and Full.

We do all our development work on a single codebase. This means that everything --the engine and all tools-- is

on one common workbench. There are **not** different versions that we work on in parallel -- that would be crazy to manage! That's why the version numbers of GATK-Lite and GATK-Full always match: if the latest GATK-Full version is numbered 2.1-13, then the latest GATK-Lite is also numbered 2.1-13.

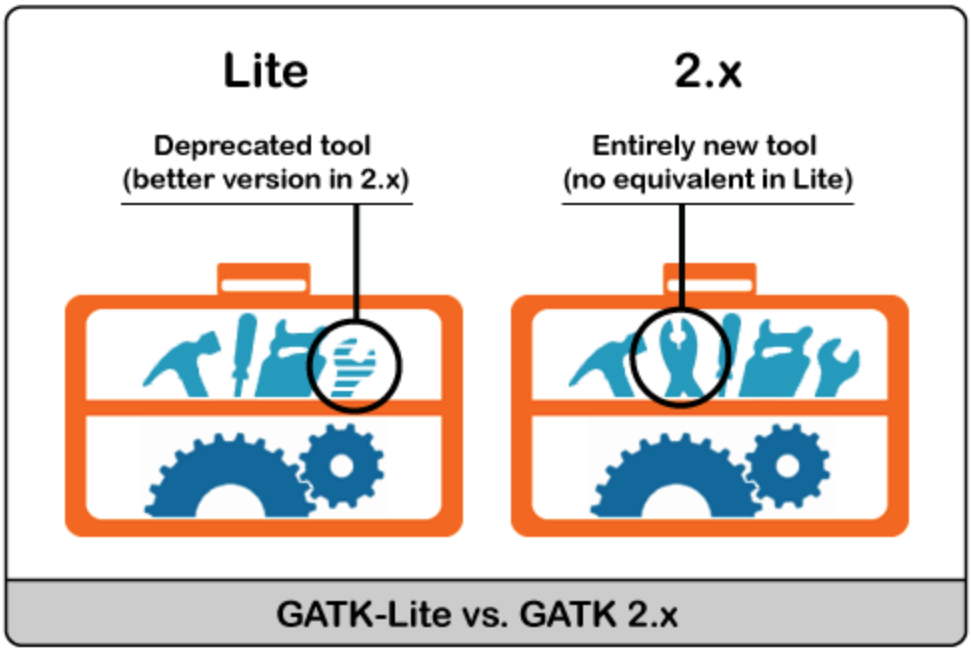
The most important consequence of this setup is that when we make improvements to the infrastructure and engine, the same improvements will end up in GATK Lite and in GATK Full. So for the purposes of power, speed and robustness of the GATK that is determined by the engine, there is no difference between them.

For the tools, it's a little more complicated -- but not much. When we "build" the GATK binaries (the `.jar` files), we put everything from the workbench into the Full build, but we only put a subset into the Lite build. Note that this Lite subset is pretty big -- it contains all the tools that were previously available in GATK 1.x versions, and always will. We also reserve the right to add previews or not-fully-featured versions of the new tools that are in Full, at our discretion, to the Lite build.

So there are two basic types of differences between the tools available in the Lite and Full builds (see picture below).

- We have a new tool that performs a brand new function (which wasn't available in GATK 1.x), and we only include it in the Full build.

- We have a tool that has some new add-on capabilities (which weren't possible in GATK 1.x); we put the tool in both the Lite and the Full build, but the add-ons are only available in the Full build.



Reprising the car analogy, GATK-Lite and GATK-Full are like two versions of the same car -- the basic version and the fully-equipped one. They both have the exact same engine, and most of the equipment (tools) is the same -- for example, they both have the same airbag system, and they both have headlights. But there are a few important differences:

- The GATK-Full car comes with a GPS (sat-nav for our UK friends), for which the Lite car has no equivalent. You could buy a portable GPS unit from a third-party store for your Lite car, but it might not be as good, and certainly not as convenient, as the Full car's built-in one.

- Both cars have windows of course, but the Full car has power windows, while the Lite car doesn't. The Lite windows can open and close, but you have to operate them by hand, which is much slower.

So, to summarize:

The underlying engine is exactly the same in both GATK-Lite and GATK-Full. Most functionalities are available in both builds, performed by the same tools. Some functionalities are available in both builds, but they are performed by different tools, and the tool in the Full build is better. New, cutting-edge functionalities are only available in the Full build, and there is no equivalent in the Lite build.

We hope this clears up some of the confusion surrounding GATK-Lite. If not, please leave a comment and we'll do our best to clarify further!

What is Map/Reduce and why are GATK tools called "walkers"? #1754

Last updated on 2013-01-14 17:35:25

Overview

One of the key challenges of working with next-gen sequence data is that input files are usually very large. We canâ€™t just make the program open the files, load all the data into memory and perform whatever analysis is needed on all of it in one go. Itâ€™s just too much work, even for supercomputers.

Instead, we make the program cut the job into smaller tasks that the computer can easily process separately. Then we have it combine the results of each step into the final result.

Map/Reduce

Map/Reduce is the technique we use to achieve this. It consists of three steps formally called `filter`, `map` and `reduce`. Letâ€™s apply it to an example case where we want to find out what is the average depth of coverage in our dataset for a certain region of the genome.

- `filter` determines what subset of the data needs to be processed in each task. In our example, the program lists all the reference positions in our region of interest.

- `map` applies the function, i.e. performs the analysis on each subset of data. In our example, for each position in the list, the program looks into the BAM file, pulls out the pileup of bases and outputs the depth of coverage at that position.

- `reduce` combines the elements in the list of results output by the `map` function. In our example, the program takes the coverage numbers that were calculated separately for all the reference positions and calculates their average, which is the final result we want.

This may seem trivial for such a simple example, but it is a very powerful method with many advantages. Among other things, it makes it relatively easy to parallelize operations, which makes the tools run much faster on large

datasets.

Walkers, filters and traversal types

All the tools in the GATK are built from the ground up to take advantage of this method. Thatâ€™s why we call them **walkers**: because they â€œwalkâ€• across the genome, getting things done.

Note that even though itâ€™s not included in the Map/Reduce techniqueâ€™s name, the `filter` step is very important. It determines what data get presented to the tool for analysis, selecting only the appropriate data for each task and discarding anything thatâ€™s not relevant. This is a key part of the Map/Reduce technique, because thatâ€™s what makes each task â€œbite-sizedâ€• enough for the computer to handle easily.

Each tool has filters that are tailored specifically for the type of analysis it performs. The filters rely on **traversal engines**, which are little programs that are designed to â€œtraverseâ€• the data (i.e. walk through the data) in specific ways.

There are three major types of traversal: **Locus Traversal**, **Read Traversal** and **Active Region Traversal**. In our interval coverage example, the toolâ€™s filter uses the **Locus Traversal** engine, which walks through the data by locus, i.e. by position along the reference genome. Because of that, the tool is classified as a **Locus Walker**. Similarly, the **Read Traversal** engine is used, youâ€™ve guessed it, by **Read Walkers**.

The GATK engine comes packed with many other ways to walk through the genome and get the job done seamlessly, but those are the ones youâ€™ll encounter most often.

Further reading

A primer on parallelism with the GATK How can I use parallelism to make GATK tools run faster?

What is a GATKReport ? #1244

Last updated on 2013-10-03 16:01:57

A GATKReport is simply a text document that contains well-formatted, easy to read representation of some tabular data. Many GATK tools output their results as GATKReports, so it's important to understand how they are formatted and how you can use them in further analyses.

Here's a simple example:

```
#:GATKReport.v1.0:2
#:GATKTable:true:2:9:%.18E:%.15f:;
#:GATKTable:ErrorRatePerCycle:The error rate per sequenced position in the reads
cycle  errorrate.61PA8.7         qualavg.61PA8.7
0      7.451835696110506E-3      25.474613284804366
1      2.362777171937477E-3      29.844949954504095
2      9.087604507451836E-4      32.875909752547310
3      5.452562704471102E-4      34.498999090081895
4      9.087604507451836E-4      35.148316651501370
```

```
5      5.452562704471102E-4      36.072234352256190
6      5.452562704471102E-4      36.121724890829700
7      5.452562704471102E-4      36.191048034934500
8      5.452562704471102E-4      36.003457059679770

#:GATKTable:false:2:3:%s:%c:;
#:GATKTable:TableName:Description
key    column
1:1000  T
1:1001  A
1:1002  C
```

This report contains two individual GATK report tables. Every table begins with a header for its metadata and then a header for its name and description. The next row contains the column names followed by the data.

We provide an R library called `gsalib` that allows you to load GATKReport files into R for further analysis. Here are four simple steps to getting `gsalib`, installing it and loading a report.

1. Start R (or open RStudio)

```
$ R

R version 2.11.0 (2010-04-22)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

2. Get the `gsalib` library from CRAN

The `gsalib` library is available on the Comprehensive R Archive Network, so you can just do:

```
> install.packages("gsalib")
```

From within R (we use RStudio for convenience).

In some cases you need to explicitly tell R where to find the library; you can do this as follows:

```
$ cat .Rprofile
.libPaths("/path/to/Sting/R/")
```

3. Load the gsalib library

```
> library(gsalib)
```

4. Finally, load the GATKReport file and have fun

```
> d = gsa.read.gatkreport("/path/to/my.gatkreport")
> summary(d)
              Length Class      Mode
CountVariants 27     data.frame list
CompOverlap   13     data.frame list
```

What should I use as known variants/sites for running tool X? #1247

Last updated on 2013-09-11 22:07:51

1. Notes on known sites

Why are they important?

Each tool uses known sites differently, but what is common to all is that they use them to help distinguish true variants from false positives, which is very important to how these tools work. If you don't provide known sites, the statistical analysis of the data will be skewed, which can dramatically affect the sensitivity and reliability of the results.

In the variant calling pipeline, the only tools that do not strictly require known sites are UnifiedGenotyper and HaplotypeCaller.

Human genomes

If you're working on human genomes, you're in luck. We provide sets of known sites in the human genome as part of our resource bundle, and we can give you specific Best Practices recommendations on which sets to use for each tool in the variant calling pipeline. See the next section for details.

Non-human genomes

If you're working on genomes of other organisms, things may be a little harder -- but don't panic, we'll try to help as much as we can. We've started a community discussion in the forum on What are the standard resources for non-human genomes? in which we hope people with non-human genomics experience will share their knowledge.

And if it turns out that there is as yet no suitable set of known sites for your organisms, here's how to make your own for the purposes of BaseRecalibration: First, do an initial round of SNP calling on your original,

unrecalibrated data. Then take the SNPs that you have the highest confidence in and use that set as the database of known SNPs by feeding it as a VCF file to the base quality score recalibrator. Finally, do a real round of SNP calling with the recalibrated data. These steps could be repeated several times until convergence. Good luck!

Some experimentation will be required to figure out the best way to find the highest confidence SNPs for use here. Perhaps one could call variants with several different calling algorithms and take the set intersection. Or perhaps one could do a very strict round of filtering and take only those variants which pass the test.

2. Recommended sets of known sites per tool

Summary table

Tool	dbSNP 129 -	- dbSNP >132 -	- Mills indels -	- 1KG indels -	- HapMap -	- Omni
RealignerTargetCreator			X	X		
IndelRealigner			X	X		
BaseRecalibrator		X	X	X		
(UnifiedGenotyper HaplotypeCaller)		X				
VariantRecalibrator		X	X		X	X
VariantEval	X					

RealignerTargetCreator and IndelRealigner

These tools require known indels passed with the `-known` argument to function properly. We use both the following files:

- Mills_and_1000G_gold_standard.indels.b37.sites.vcf
- 1000G_phase1.indels.b37.vcf (currently from the 1000 Genomes Phase I indel calls)

BaseRecalibrator

This tool requires known SNPs and indels passed with the `-knownSites` argument to function properly. We use all the following files:

- The most recent dbSNP release (build ID > 132)
- Mills_and_1000G_gold_standard.indels.b37.sites.vcf
- 1000G_phase1.indels.b37.vcf (currently from the 1000 Genomes Phase I indel calls)

UnifiedGenotyper / HaplotypeCaller

These tools do NOT require known sites, but if SNPs are provided with the `-dbsnp` argument they will use them for variant annotation. We use this file:

- The most recent dbSNP release (build ID > 132)

VariantRecalibrator

For VariantRecalibrator, please see the FAQ article on VQSR training sets and arguments.

VariantEval

This tool requires known SNPs passed with the `-dbsnp` argument to function properly. We use the following file:

- A version of dbSNP subsetted to only sites discovered in or before dbSNP BuildID 129, which excludes the impact of the 1000 Genomes project and is useful for evaluation of dbSNP rate and Ti/Tv values at novel sites.

What's in the resource bundle and how can I get it? #1213

Last updated on 2013-03-25 18:21:28

1. Obtaining the bundle

Inside of the Broad, the latest bundle will always be available in:

```
/humgen/gsa-hpprojects/GATK/bundle/current
```

with a subdirectory containing for each reference sequence and associated data files.

External users can download these files (or corresponding .gz versions) from the GSA FTP Server in the directory bundle. Gzipped files should be unzipped before attempting to use them. Note that there is no "current link" on the FTP; users should download the highest numbered directory under current (this is the most recent data set).

2. b37 Resources: the Standard Data Set

- Reference sequence (standard 1000 Genomes fasta) along with fai and dict files
- dbSNP in VCF. This includes two files:
- The most recent dbSNP release
- This file subsetted to only sites discovered in or before dbSNPBuildID 129, which excludes the impact of the 1000 Genomes project and is useful for evaluation of dbSNP rate and Ti/Tv values at novel sites.

- HapMap genotypes and sites VCFs
- OMNI 2.5 genotypes for 1000 Genomes samples, as well as sites, VCF
- The current best set of known indels to be used for local realignment (note that we don't use dbSNP for this anymore); use both files:
  - 1000G_phase1.indels.b37.vcf (currently from the 1000 Genomes Phase I indel calls)
  - Mills_and_1000G_gold_standard.indels.b37.sites.vcf

- A large-scale standard single sample BAM file for testing:
  - NA12878.HiSeq.WGS.bwa.cleaned.recal.hg19.20.bam containing ~64x reads of NA12878 on chromosome

20

- The results of the latest UnifiedGenotyper with default arguments run on this data set (NA12878.HiSeq.WGS.bwa.cleaned.recal.hg19.20.vcf)

Additionally, these files all have supplementary indices, statistics, and other QC data available.

3. hg18 Resources: lifted over from b37

Includes the UCSC-style hg18 reference along with all lifted over VCF files. The refGene track and BAM files are not available. We only provide data files for this genome-build that can be lifted over "easily" from our master b37 repository. Sorry for whatever inconvenience that this might cause.

Also includes a chain file to lift over to b37.

4. b36 Resources: lifted over from b37

Includes the 1000 Genomes pilot b36 formated reference sequence (human_b36_both.fasta) along with all lifted over VCF files. The refGene track and BAM files are not available. We only provide data files for this genome-build that can be lifted over "easily" from our master b37 repository. Sorry for whatever inconvenience that this might cause.

Also includes a chain file to lift over to b37.

5. hg19 Resources: lifted over from b37

Includes the UCSC-style hg19 reference along with all lifted over VCF files.

Where can I get more information about next-generation sequencing concepts and terms? #1321

We know this field can be confusing or even overwhelming to newcomers, and getting to grips with a large and varied toolkit like the GATK can be a big challenge. We have produce a presentation that we hope will help you review all the background information that you need to know in order to use the GATK:

- Introduction to NGS Analysis: all you need to know to use the GATK: slides and video

In addition, the following links feature a lot of useful educational material about concepts and terminology related to next-generation sequencing:

- DNA sequencing (Wikipedia)

A basic review of the sequencing process.

- Sequencing technologies, the next generation, (M. Metzker, Nature Reviews - Genetics)

An excellent, detailed overview of the myriad next-gen sequencing methdologies.

- Next-generation sequencing: adjusting to data overload (M. Baker, Nature Methods)

A nice piece explaining the problems inherent in trying to analyze terabytes of data. The GATK addresses this issue by requiring all datasets be in reference order, so only small chunks of the genome need to be in memory at once, as explained here.

- Primer on NGS analysis, from Broad Institute Primers in Medical Genetics

Which data processing steps should I do per-lane vs. per-sample? #3060

Last updated on 2013-09-07 14:04:11

Note that there are many possible ways to achieve a similar result; here we present the way we think gives the best combination of efficiency and quality. This assumes that you are dealing with one or more samples, and each of them was sequenced on one or more lanes.

Let's say we have this example data:

- sample1_lane1.fq
- sample1_lane2.fq
- sample2_lane1.fq
- sample2_lane2.fq

1. Run all core steps per-lane once

At the basic level, all pre-processing steps are meant to be performed per-lane. Assuming that you received one FASTQ file per lane of sequence data, just run each file through each pre-processing step individually: map & dedup -> realign -> recal.

The example data becomes:

- sample1_lane1.dedup.realn.recal.bam
- sample1_lane2.dedup.realn.recal.bam
- sample2_lane1.dedup.realn.recal.bam
- sample2_lane2.dedup.realn.recal.bam

2. Merge lanes per sample

Once you have pre-processed each lane individually, you merge lanes belonging to the same sample into a single BAM file.

The example data becomes:

- sample1.merged.bam

- sample2.merged.bam

3. Per-sample refinement

You can increase the quality of your results by performing an extra round of dedupping and realignment, this time at the sample level. It is not absolutely required and will increase your computational costs, so it's up to you to decide whether you want to do it on your data, but that's how we do it internally at Broad.

The example data becomes:

- sample1.merged.dedup.realn.bam
- sample2.merged.dedup.realn.bam

This gets you two big wins:

- Dedupping per-sample eliminates PCR duplicates across all lanes in addition to optical duplicates (which are by definition only per-lane)
- Realigning per-sample means that you will have consistent alignments across all lanes within a sample.

People often ask also if it's worth the trouble to try realigning across all samples in a cohort. The answer is almost always no, unless you have very shallow coverage. The problem is that while it would be lovely to ensure consistent alignments around indels across all samples, the computational cost gets too ridiculous too fast. That being said, for contrastive calling projects -- such as cancer tumor/normals -- we do recommend realigning both the tumor and the normal together in general to avoid slight alignment differences between the two tissue types.

Finally, why not do base recalibration across lanes or across samples? Well, by definition there is no sense in trying to recalibrate across lanes, since the purpose of this processing step is to compensate for the errors made by the machine during sequencing, and the lane is the base unit of the sequencing machine. That said, don't worry if you find yourself needing to recalibrate a BAM file with the lanes already merged -- the GATK's BaseRecalibrator is read group-aware, which means that it will identify separate lanes as such even if they are in the same BAM file, and it will always process them separately.

Which datasets should I use for reviewing or benchmarking purposes? #1292

Last updated on 2013-03-25 21:51:48

New WGS and WEx CEU trio BAM files

We have sequenced at the Broad Institute and released to the 1000 Genomes Project the following datasets for the three members of the CEU trio (NA12878, NA12891 and NA12892):

- WEx (150x) sequence
- WGS (~60x) sequence

This is better data to work with than the original DePristo et al. BAMs files, so we recommend you download and analyze these files if you are looking for complete, large-scale data sets to evaluate the GATK or other tools.

Here's the rough library properties of the BAMs:

Sample	Library	Insert Size	Read Length	Depth	Origin
NA12878 WGS	WGS	400 bp	101 bp	64x	DePristo (2011)
	WGS wide	400 bp	101 bp	7x	New sequencing
	fragment	155 bp	101 bp	45x	Gnerre (2011)
	jumping	2.3-2.8 kbp	101 bp	50x	Gnerre (2011)
	fosmid	35 kbp	101 bp	1x	Gnerre (2011)
NA12891 WGS	WGS	400 bp	101 bp	37x	New sequencing
	WGS wide	400 bp	101 bp	34x	New sequencing
NA12892 WGS	WGS	400 bp	101 bp	37x	New sequencing
	WGS wide	400 bp	101 bp	34x	New sequencing
NA12878 WEX	WEX standard	155 bp	76 bp	~150x	New sequencing
NA12891 WEX	WEX standard	155 bp	76 bp	~150x	New sequencing
NA12892 WEX	WEX standard	155 bp	76 bp	~150x	New sequencing

These data files can be downloaded from the 1000 Genomes DCC

NA12878 Datasets from DePristo et al. (2011) Nature Genetics

Here are the datasets we used in the GATK paper cited below.

DePristo M, Banks E, Poplin R, Garimella K, Maguire J, Hartl C, Philippakis A, del Angel G, Rivas MA, Hanna M, McKenna A, Fennell T, Kernytsky A, Sivachenko A, Cibulskis K, Gabriel S, Altshuler D and Daly, M (2011). A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature Genetics. **43:**491-498.

Some of the BAM and VCF files are currently hosted by the NCBI:
ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/working/20101201_cg_NA12878/

- NA12878.hiseq.wgs.bwa.recal.bam -- BAM file for NA12878 HiSeq whole genome
- NA12878.hiseq.wgs.bwa.raw.bam Raw reads (in BAM format, see below)
- NA12878.ga2.exome.maq.recal.bam -- BAM file for NA12878 GenomeAnalyzer II whole exome (hg18)
- NA12878.ga2.exome.maq.raw.bam Raw reads (in BAM format, see below)

- NA12878.hiseq.wgs.vcf.gz -- SNP calls for NA12878 HiSeq whole genome (hg18)

- NA12878.ga2.exome.vcf.gz -- SNP calls for NA12878 GenomeAnalyzer II whole exome (hg18)

- BAM files for CEU + NA12878 whole genome (b36). These are the standard BAM files for the 1000 Genomes pilot CEU samples plus a 4x downsampled version of NA12878 from the pilot 2 data set, available in the DePristoNatGenet2011 directory of the GSA FTP Server

- SNP calls for CEU + NA12878 whole genome (b36) are available in the DePristoNatGenet2011 directory of the GSA FTP Server

- Crossbow comparison SNP calls are available in the DePristoNatGenet2011 directory of the GSA FTP Server as crossbow.filtered.vcf. The raw calls can be viewed by ignoring the `FILTER` field status

- whole_exome_agilent_designed_120.Homo_sapiens_assembly18.targets.interval_list `--targets` used in the analysis of the exome capture data

Please note that we have not collected the indel calls for the paper, as these are only used for filtering SNPs near indels. If you want to call accurate indels, please use the new GATK indel caller in the Unified Genotyper.

Warnings

Both the GATK and the sequencing technologies have improved significantly since the analyses performed in this paper.

- If you are conducting a review today, we would recommend that the newest version of the GATK, which performs much better than the version described in the paper. Moreover, we would also recommend one use the newest version of Crossbow as well, in case they have improved things. The GATK calls for NA12878 from the paper (above) will give one a good idea what a good call set looks like whole-genome or whole-exome.

- The data sets used in the paper are no longer state-of-the-art. The WEx BAM is GAII data aligned with MAQ on hg18, but a state-of-the-art data set would use HiSeq and BWA on hg19. Even the 64x HiSeq WG data set is already more than one year old. For a better assessment, we would recommend you use a newer data set for these samples, if you have the capacity to generate it. This applies less to the WG NA12878 data, which is pretty good, but the NA12878 WEx from the paper is nearly 2 years old now and notably worse than our most recent data sets.

Obviously, this was an annoyance for us as well, as it would have been nice to use a state-of-the-art data set for the WEx. But we decided to freeze the data used for analysis to actually finish this paper.

How do I get the raw FASTQ file from a BAM?

If you want the raw, machine output for the data analyzed in the GATK framework paper, obtain the raw BAM files above and convert them from SAM to FASTQ using the Picard tool SamToFastq.

Why are some of the annotation values different with VariantAnnotator compared to UG or HC? #1550

As featured in this forum question.

Two main things account for these kinds of differences, both linked to default behaviors of the tools:

- The tools downsample to different depths of coverage
- The tools apply different read filters

In both cases, you can end up looking at different sets or numbers of reads, which causes some of the annotation values to be different. It's usually not a cause for alarm. Remember that many of these annotations should be interpreted relatively, not absolutely.

Why didn't the UG or HC call my SNP? I can see it right there in IGV! #1235

Last updated on 2013-08-23 22:02:20

Just because something looks like a SNP in IGV doesn't mean that it is of high quality. We are extremely confident in the genotype likelihoods calculations in the Unified Genotyper (especially for SNPs) and in the HaplotypeCaller (for all variants including indels). So, before you post this issue in our support forum, please do a little bit of investigation on your own, as follows.

To diagnose what is happening, you should take a look at the pileup of bases at the position in question. It is very important for you to look at the underlying data here.

Here is a checklist of questions you should ask yourself:

- **How many overlapping deletions are there at the position?**

The genotyper ignores sites if there are too many overlapping deletions. This value can be set using the `--max_deletion_fraction` argument (see the UG's documentation page to find out what is the default value for this argument), but be aware that increasing it could affect the reliability of your results.

- **What do the base qualities look like for the non-reference bases?**

Remember that there is a minimum base quality threshold and that low base qualities mean that the sequencer assigned a low confidence to that base. If your would-be SNP is only supported by low-confidence bases, it is probably a false positive.

Keep in mind that the depth reported in the VCF is the unfiltered depth. You may think you have good coverage at that site, but the Unified Genotyper ignores bases if they don't look good, so actual coverage seen by the UG may be lower than you think.

- **What do the mapping qualities look like for the reads with the non-reference bases?**

A base's quality is capped by the mapping quality of its read. The reason for this is that low mapping qualities mean that the aligner had little confidence that the read is mapped to the correct location in the genome. You may be seeing mismatches because the read doesn't belong there -- you may be looking at the sequence of some other locus in the genome!

Keep in mind also that reads with mapping quality 255 ("unknown") are ignored.

- **Are there a lot of alternate alleles?**

By default the UG will only consider a certain number of alternate alleles. This value can be set using the `--max_alternate_alleles` argument (see the UG's documentation page to find out what is the default value for this argument). Note however that genotyping sites with many alternate alleles is both CPU and memory intensive and it scales exponentially based on the number of alternate alleles. Unless there is a good reason to change the default value, we highly recommend that you not play around with this parameter.

- **Are you working with SOLiD data?**

SOLiD alignments tend to have reference bias and it can be severe in some cases. Do the SOLiD reads have a lot of mismatches (no-calls count as mismatches) around the the site? If so, you are probably seeing false positives.

Tutorials

This section contains tutorials that will teach you step-by-step how to use GATK tools and how to solve common problems.

(howto) Apply hard filters to a call set #2806

Posted on 2013-06-17 22:48:43

Objective

Apply hard filters to a variant callset that is too small for VQSR or for which truth/training sets are not available.

Prerequisites

- TBD

Steps

- Extract the SNPs from the call set
- Determine parameters for filtering SNPs
- Apply the filter to the SNP call set
- Extract the Indels from the call set
- Determine parameters for filtering SNPs
- Apply the filter to the Indel call set

---

1. Extract the SNPs from the call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T SelectVariants \
     -R reference.fa \
     -V raw_variants.vcf \
     -L 20 \
     -selectType SNP \
     -o raw_snps.vcf
```

Expected Result

This creates a VCF file called `raw_snps.vcf`, containing just the SNPs from the original file of raw variants.

---

2. Determine parameters for filtering SNPs

SNPs matching any of these conditions will be considered bad and filtered out, i.e. marked `FILTER` in the output VCF file. The program will specify which parameter was chiefly responsible for the exclusion of the SNP using the culprit annotation. SNPs that do not match any of these conditions will be considered good and marked `PASS` in the output VCF file.

- QualByDepth (QD) 2.0

This is the variant confidence (from the `QUAL` field) divided by the unfiltered depth of non-reference samples.

- FisherStrand (FS) 60.0

Phred-scaled p-value using Fisherâ€™s Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- RMSMappingQuality (MQ) 40.0

This is the Root Mean Square of the mapping quality of the reads across all samples.

- HaplotypeScore 13.0

This is the consistency of the site with two (and only two) segregating haplotypes. Note that this is not applicable for calls made using the UnifiedGenotyper on non-diploid organisms.

- MappingQualityRankSumTest (MQRankSum) 12.5

This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for mapping qualities (reads with ref bases vs. those with the alternate allele). Note that the mapping quality rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

- ReadPosRankSumTest (ReadPosRankSum) 8.0

This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

---

3. Apply the filter to the SNP call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T VariantFiltration \
     -R reference.fa \
     -V raw_snps.vcf \
     --filterExpression "QD < 2.0 || FS > 60.0 || MQ < 40.0 || HaplotypeScore > 13.0 ||
MappingQualityRankSum < -12.5 || ReadPosRankSum < -8.0" \
     --filterName "my_snp_filter" \
     -o filtered_snps.vcf
```

Expected Result

This creates a VCF file called `filtered_snps.vcf`, containing all the original SNPs from the `raw_snps.vcf` file, but now the SNPs are annotated with either PASS or `FILTER` depending on whether or not they passed the filters.

For SNPs that failed the filter, the variant annotation also includes the name of the filter. That way, if you apply several different filters (simultaneously or sequentially), you can keep track of which filter(s) each SNP failed, and later you can retrieve specific subsets of your calls using the SelectVariants tool. To learn more about composing different types of filtering expressions and retrieving subsets of variants using SelectVariants, please see the online GATK documentation.

---

4. Extract the Indels from the call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T SelectVariants \
     -R reference.fa \
     -V raw_HC_variants.vcf \
     -L 20 \
     -selectType INDEL \
     -o raw_indels.vcf
```

Expected Result

This creates a VCF file called `raw_indels.vcf`, containing just the Indels from the original file of raw variants.

---

5. Determine parameters for filtering Indels.

Indels matching any of these conditions will be considered bad and filtered out, i.e. marked `FILTER` in the output

VCF file. The program will specify which parameter was chiefly responsible for the exclusion of the indel using the culprit annotation. Indels that do not match any of these conditions will be considered good and marked PASS in the output VCF file.

- QualByDepth (QD) 2.0

This is the variant confidence (from the QUAL field) divided by the unfiltered depth of non-reference samples.

- FisherStrand (FS) 200.0

Phred-scaled p-value using Fisherâ€™s Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- ReadPosRankSumTest (ReadPosRankSum) 20.0

This is the u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles, i.e. this will only be applied to heterozygous calls.

---

6. Apply the filter to the Indel call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T VariantFiltration \
     -R reference.fa \
     -V raw_indels.vcf \
     --filterExpression "QD < 2.0 || FS > 200.0 || ReadPosRankSum < -20.0" \
     --filterName "my_indel_filter" \
     -o filtered_indels.vcf
```

Expected Result

This creates a VCF file called `filtered_indels.vcf`, containing all the original Indels from the `raw_indels.vcf` file, but now the Indels are annotated with either PASS or FILTER depending on whether or not they passed the filters.

For Indels that failed the filter, the variant annotation also includes the name of the filter. That way, if you apply several different filters (simultaneously or sequentially), you can keep track of which filter(s) each Indel failed, and later you can retrieve specific subsets of your calls using the SelectVariants tool. To learn more about composing different types of filtering expressions and retrieving subsets of variants using SelectVariants, please see the online GATK documentation.

(howto) Call variants on a diploid genome with the HaplotypeCaller #2803

Last updated on 2013-10-03 18:40:59

Objective

Call variants on a diploid genome with the HaplotypeCaller, producing a raw (unfiltered) VCF.

Prerequisites

- TBD

Steps

- Determine the basic parameters of the analysis

- Call variants in your sequence data

1. Determine the basic parameters of the analysis

If you do not specify these parameters yourself, the program will use default values. However we recommend that you set them explicitly because it will help you understand how the results are bounded and how you can modify the program's behavior.

- Genotyping mode (â€“`genotyping_mode`)

This specifies how we want the program to determine the alternate alleles to use for genotyping. In the default `DISCOVERY` mode, the program will choose the most likely alleles out of those it sees in the data. In `GENOTYPE_GIVEN_ALLELES` mode, the program will only use the alleles passed in from a VCF file (using the `-alleles` argument). This is useful if you just want to determine if a sample has a specific genotype of interest and you are not interested in other alleles.

- Emission confidence threshold (â€“`stand_emit_conf`)

This is the minimum confidence threshold (phred-scaled) at which the program should emit sites that appear to be possibly variant.

- Calling confidence threshold (â€“`stand_call_conf`)

This is the minimum confidence threshold (phred-scaled) at which the program should emit variant sites as called. If a site's associated genotype has a confidence score lower than the calling threshold, the program will emit the site as filtered and will annotate it as LowQual. This threshold separates high confidence calls from low confidence calls.

The terms called and filtered are tricky because they can mean different things depending on context. In ordinary language, people often say a site was called if it was emitted as variant. But in the GATK's technical language,

saying a site was called means that that site passed the confidence threshold test. For filtered, it's even more confusing, because in ordinary language, when people say that sites were filtered, they usually mean that those sites successfully passed a filtering test. However, in the GATK's technical language, the same phrase (saying that sites were filtered) means that those sites failed the filtering test. In effect, it means that those would be filtered out if the filter was used to actually remove low-confidence calls from the callset, instead of just tagging them. In both cases, both usages are valid depending on the point of view of the person who is reporting the results. So it's always important to check what is the context when interpreting results that include these terms.

---

2. Call variants in your sequence data

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T HaplotypeCaller \
     -R reference.fa \
     -I preprocessed_reads.bam \  # can be reduced or not
     -L 20 \
     --genotyping_mode DISCOVERY \
     -stand_emit_conf 10 \
     -stand_call_conf 30 \
     -o raw_variants.vcf
```

Expected Result

This creates a VCF file called `raw_variants.vcf`, containing all the sites that the HaplotypeCaller evaluated to be potentially variant. Note that this file contains both SNPs and Indels.

Although you now have a nice fresh set of variant calls, the variant discovery stage is not over. The distinctions made by the caller itself between low-confidence calls and the rest is very primitive, and should not be taken as a definitive guide for filtering. The GATK callers are designed to be very lenient in calling variants, so it is extremely important to apply one of the recommended filtering methods (variant recalibration or hard-filtering), in order to move on to downstream analyses with the highest-quality call set possible.

(howto) Call variants on a non-diploid genome with the UnifiedGenotyper #2804

Last updated on 2013-06-19 17:32:04

Objective

Call variants on a haploid genome with the UnifiedGenotyper, producing a raw (unfiltered) VCF.

Prerequisites

- TBD

Steps

- Determine the basic parameters of the analysis
- Call variants in your sequence data

1. Determine the basic parameters of the analysis

If you do not specify these parameters yourself, the program will use default values. However we recommend that you set them explicitly because it will help you understand how the results are bounded and how you can modify the program's behavior.

- Ploidy (`-ploidy`)

In its basic use, this is the ploidy (number of chromosomes) per sample. By default it is set to 2, to process diploid organisms' genomes, but it can be set to any other desired value, starting at 1 for haploid organisms, and up for polyploids. This argument can also be used to handle pooled data. For that purpose, you'll need to set `-ploidy` to `Number of samples in each pool * Sample Ploidy`. There is no fixed upper limit, but keep in mind that high-level ploidy will increase processing times since the calculations involved are more complex. For full details on how to process pooled data, see Eran et al. (in preparation).

- Genotype likelihood model (`-glm`)

This is the model that the program will use to calculate the genotype likelihoods. By default, it is set to `SNP`, but it can also be set to `INDEL` or `BOTH`. If set to `BOTH`, both SNPs and Indels will be called in the same run and be output to the same variants file.

- Emission confidence threshold (`â€“stand_emit_conf`)

This is the minimum confidence threshold (phred-scaled) at which the program should emit sites that appear to be possibly variant.

- Calling confidence threshold (`â€“stand_call_conf`)

This is the minimum confidence threshold (phred-scaled) at which the program should emit variant sites as called. If a site's associated genotype has a confidence score lower than the calling threshold, the program will emit the site as filtered and will annotate it as LowQual. This threshold separates high confidence calls from low confidence calls.

The terms called and filtered are tricky because they can mean different things depending on context. In ordinary language, people often say a site was called if it was emitted as variant. But in the GATK's technical language, saying a site was called means that that site passed the confidence threshold test. For filtered, it's even more confusing, because in ordinary language, when people say that sites were filtered, they usually mean that those

sites successfully passed a filtering test. However, in the GATK's technical language, the same phrase (saying that sites were filtered) means that those sites failed the filtering test. In effect, it means that those would be filtered out if the filter was used to actually remove low-confidence calls from the callset, instead of just tagging them. In both cases, both usages are valid depending on the point of view of the person who is reporting the results. So it's always important to check what is the context when interpreting results that include these terms.

---

2. Call variants in your sequence data

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T UnifiedGenotyper \
     -R haploid_reference.fa \
     -I haploid_reads.bam \
     -L 20 \
     -ploidy 1
     --glm BOTH \
     --stand\_call\_conf 30 \
     --stand\_emit\_conf 10 \
     -o raw_haploid_variants.vcf
```

This creates a VCF file called `raw_haploid_variants.vcf`, containing all the sites that the UnifiedGenotyper evaluated to be potentially variant.

Although you now have a nice fresh set of variant calls, the variant discovery stage is not over. The distinctions made by the caller itself between low-confidence calls and the rest is very primitive, and should not be taken as a definitive guide for filtering. The GATK callers are designed to be very lenient in calling variants, so it is extremely important to apply one of the recommended filtering methods (variant recalibration or hard-filtering), in order to move on to downstream analyses with the highest-quality call set possible.

(howto) Compress read data with ReduceReads #2802

Last updated on 2013-06-17 21:25:02

Objective

Compress the read data in order to minimize file sizes, which facilitates massively multisample processing.

Prerequisites

- TBD

Steps

- Compress your sequence data

1. Compress your sequence data

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T ReduceReads \
     -R reference.fa \
     -I recal_reads.bam \
     -L 20 \
     -o reduced_reads.bam
```

Expected Result

This creates a file called `reduced_reads.bam` containing only the sequence information that is essential for calling variants.

Note that ReduceReads is not meant to be run on multiple samples at once. If you plan on merging your sample bam files, you should run ReduceReads on individual samples before doing so.

(howto) Fix a badly formatted BAM #2909

Last updated on 2013-07-03 00:54:26

Objective

Fix a BAM that is not indexed or not sorted, has not had duplicates marked, or is lacking read group information. These steps can be performed independently of each other but this order is recommended.

Prerequisites

- Installed Picard tools

Steps

- Sort the aligned reads by coordinate order

- Mark duplicates

- Add read group information

- Index the BAM file

Note

You may ask, is all of this really necessary? The GATK is notorious for imposing strict formatting guidelines and requiring the presence of information such as read groups that other software packages do not require. Although this represents a small additional processing burden upfront, the downstream benefits are numerous, including the ability to process library data individually, and significant gains in speed and parallelization options.

1. Sort the aligned reads by coordinate order

Action

Run the following Picard command:

```
java -jar SortSam.jar \
    INPUT=unsorted_reads.bam \
    OUTPUT=sorted_reads.bam \
    SORT_ORDER=coordinate
```

Expected Results

This creates a file called `sorted_reads.bam` containing the aligned reads sorted by coordinate.

---

2. Mark duplicate reads

Action

Run the following Picard command:

```
java -jar MarkDuplicates.jar \
    INPUT=sorted_reads.bam \
    OUTPUT=dedup_reads.bam
```

Expected Results

This creates a file called `dedup_reads.bam` with the same content as the input file, except that any duplicate reads are marked as such.

More details

During the sequencing process, the same DNA molecules can be sequenced several times. The resulting duplicate reads are not informative and should not be counted as additional evidence for or against a putative variant. The duplicate marking process (sometimes called dedupping in bioinformatics slang) identifies these reads as such so that the GATK tools know to ignore them.

---

3. Add read group information

Action

Run the following Picard command:

```
java -jar AddOrReplaceReadGroups.jar  \
```

```
    INPUT=dedup_reads.bam \
    OUTPUT=addrg_reads.bam \
    RGID=group1 RGLB= lib1 RGPL=illumina RGPU=unit1 RGSM=sample1
```

Expected Results

This creates a file called `addrg_reads.bam` with the same content as the input file, except that the reads will now have read group information attached.

---

4. Index the BAM file

Action

Run the following Picard command:

```
java -jar BuildBamIndex \
    INPUT=addrg_reads.bam
```

Expected Results

This creates an index file called `addrg_reads.bai`, which is ready to be used in the Best Practices workflow.

Since Picard tools do not systematically create an index file when they output a new BAM file (unlike GATK tools, which will always output indexed files), it is best to keep the indexing step for last.

(howto) Install all software packages required to follow the GATK Best Practices#2899

Last updated on 2013-10-03 15:46:39

Objective

Install all software packages required to follow the GATK Best Practices.

Prerequisites

To follow these instructions, you will need to have a basic understanding of the meaning of the following words and command-line operations. If you are unfamiliar with any of the following, you should consult a more experienced colleague or your systems administrator if you have one. There are also many good online tutorials you can use to learn the necessary notions.

- Basic Unix environment commands
- Binary / Executable
- Compiling a binary
- Adding a binary to your path

- Command-line shell, terminal or console
- Software library

You will also need to have access to an ANSI compliant C++ compiler and the tools needed for normal compilations (make, shell, the standard library, tar, gunzip). These tools are usually pre-installed on Linux/Unix systems. On MacOS X, you may need to install the MacOS Xcode tools. See https://developer.apple.com/xcode/ for relevant information and software downloads.

Starting with version 2.6, the GATK requires Java Runtime Environment version 1.7. All Linux/Unix and MacOS X systems should have a JRE pre-installed, but the version may vary. To test your Java version, run the following command in the shell:

```
java -version
```

This should return a message along the lines of â€•java version 1.7.0_25â€• as well as some details on the Runtime Environment (JRE) and Virtual Machine (VM). If you have a version other than 1.7.x, be aware that you may run into trouble with some of the more advanced features of the Picard and GATK tools. The simplest solution is to install an additional JRE and specify which you want to use at the command-line. To find out how to do so, you should seek help from your systems administrator.

Software packages

- BWA
- SAMtools
- HTSlib
- Picard
- Genome Analysis Toolkit (GATK)
- RStudio IDE and R libraries ggplot2 and gsalib

1. BWA

Read the overview of the BWA software on the BWA project homepage, then download the latest version of the software package.

- Installation

Unpack the tar file using:

```
tar xvzf bwa-0.7.5a.tar.bz2
```

This will produce a directory called `bwa-0.7.5a` containing the files necessary to compile the BWA binary. Move to this directory and compile using:

```
cd bwa-0.7.5a
```

```
make
```

The compiled binary is called `bwa`. You should find it within the same folder (`bwa-0.7.5a` in this example). You may also find other compiled binaries; at time of writing, a second binary called `bwamem-lite` is also included. You can disregard this file for now. Finally, just add the BWA binary to your path to make it available on the command line. This completes the installation process.

- Testing

Open a shell and run:

```
bwa
```

This should print out some version and author information as well as a list of commands. As the **Usage** line states, to use BWA you will always build your command lines like this:

```
bwa <command> [options]
```

This means you first make the call to the binary (`bwa`), then you specify which command (method) you wish to use (e.g. `index`) then any options (i.e. arguments such as input files or parameters) used by the program to perform that command.

---

2. SAMtools

Read the overview of the SAMtools software on the SAMtools project homepage, then download the latest version of the software package.

- Installation

Unpack the tar file using:

```
tar xvzf samtools-0.1.19.tar.bz2
```

This will produce a directory called `samtools-0.1.19` containing the files necessary to compile the SAMtools binary. Move to this directory and compile using:

```
cd samtools-0.1.19
make
```

The compiled binary is called `samtools`. You should find it within the same folder (`samtools-0.1.19` in this example). Finally, add the SAMtools binary to your path to make it available on the command line. This completes the installation process.

- Testing

Open a shell and run:

```
samtools
```

This should print out some version information as well as a list of commands. As the **Usage** line states, to use SAMtools you will always build your command lines like this:

```
samtools <command> [options]
```

This means you first make the call to the binary (`samtools`), then you specify which command (method) you wish to use (e.g. `index`) then any options (i.e. arguments such as input files or parameters) used by the program to perform that command. This is the same convention as used by BWA.

---

3. HTSlib

Read the overview of the HTSlib software on the HTSlib project homepage, then download the latest version of the software package.

- Installation

Unpack the tar file using:

```
tar xjf htslib-master.zip
```

This will produce a directory called `htslib-master` containing the files necessary to compile the HTSlib binary. Move to this directory and compile using:

```
cd htslib-master
make
```

The compiled binary is called `htscmd`. You should find it within the same folder (`htslib-master` in this example). Finally, add the HTSlib binary to your path to make it available on the command line. This completes the installation process.

- Testing

Open a shell and run:

```
htscmd
```

This should print out some version information as well as a list of commands. As the **Usage** line states, to use HTSlib you will always build your command lines like this:

```
htscmd <command> [options]
```

This means you first make the call to the binary (`htscmd`), then you specify which command (method) you wish to use (e.g. `index`) then any options (`i.e.` arguments such as input files or parameters) used by the program to perform that command. This is the same convention as used by BWA and SAMtools.

---

4. Picard

Read the overview of the Picard software on the Picard project homepage, then download the latest version of the software package.

- Installation

Unpack the zip file using:

```
tar xjf picard-tools-1.94.zip
```

This will produce a directory called `picard-tools-1.94` containing the Picard jar files. Picard tools are distributed as pre-compiled Java executables (jar files) so there is no need to compile them. Finally, add the Picard directory to your path to make the tools available on the command line. This completes the installation process.

- Testing

Open a shell and run:

```
java -jar AddOrReplaceReadGroups.jar -h
```

This should print out some version and usage information about the `AddOrReplaceReadGroups.jar` tool. At this point you will have noticed an important difference between BWA and Picard tools. To use BWA, we called on the BWA program and specified which of its internal tools we wanted to apply. To use Picard, we called on Java itself as the main program, then specified which jar file to use, knowing that one jar file = one tool. This applies to all Picard tools; to use them you will always build your command lines like this:

```
java -jar <ToolName.jar> [options]
```

Next we will see that GATK tools are called in yet another way. The reasons for how tools in a given software package are organized and invoked are largely due to the preferences of the software developers. They generally do not reflect strict technical requirements, although they can have an effect on speed and efficiency.

---

5. Genome Analysis Toolkit (GATK)

Hopefully if you're reading this, you're already acquainted with the purpose of the GATK, so go ahead and download the latest version of the software package.

In order to access the downloads, you need to register for a free account on the GATK support forum. You will also need to read and accept the license agreement before downloading the GATK software package. Note that if you intend to use the GATK for commercial purposes, you will need to purchase a license from our commercial partner, Appistry. See Appistry's GATK FAQ page for an overview of the commercial licensing conditions.

- Installation

Unpack the tar file using:

```
tar xjf GenomeAnalysisTK-2.6-4.tar.bz2
```

This will produce a directory called `GenomeAnalysisTK-2.6-4-g3e5ff60` containing the GATK jar file, which is called `GenomeAnalysisTK.jar`, as well as a directory of example files called `resources`. GATK tools are distributed as a single pre-compiled Java executable so there is no need to compile them. Finally, add the GATK directory to your path to make the tools available on the command line. This completes the installation process.

- Testing

Open a shell and run:

```
java -jar GenomeAnalysisTK.jar -h
```

This should print out some version and usage information, as well as a list of the tools included in the GATK. As the **Usage** line states, to use GATK you will always build your command lines like this:

```
java -jar GenomeAnalysisTK.jar -T <ToolName> [arguments]
```

This means you first make the call to Java itself as the main program, then specify the `GenomeAnalysisTK.jar` file, then specify which tool you want, and finally you pass whatever other arguments (input files, parameters etc.) are needed for the analysis.

So this way of calling the program and selecting which tool to run is a little like a hybrid of how we called BWA and how we called Picard tools. To put it another way, if BWA is a standalone game device that comes preloaded with several games, Picard tools are individual game cartridges that plug into the Java console, and GATK is a single cartridge that also plugs into the Java console but contains many games.

---

6. RStudio IDE and R libraries ggplot2 and gsalib

Download the latest version of RStudio IDE. The webpage should automatically detect what platform you are running on and recommend the version most suitable for your system.

- Installation

Follow the installation instructions provided. Binaries are provided for all major platforms; typically they just need to be placed in your Applications (or Programs) directory. Open RStudio and type the following command in the console window:

```
install.packages("ggplot2")
```

This will download and install the ggplot2 library as well as any other library packages that ggplot2 depends on for its operation.

Now do the same thing to install the gsalib library:

```
install.packages("gsalib")
```

This will download and install the gsalib library.

(howto) Map and mark duplicates #2799

Last updated on 2013-09-26 19:33:51

Objective

Map the read data to the reference and mark duplicates.

Prerequisites

- TBD

Steps

- Identify read group information
- Generate a SAM file containing aligned reads
- Convert to BAM, sort and mark duplicates

---

1. Identify read group information

The read group information is key for downstream GATK functionality. The GATK will not work without a read group tag. Make sure to enter as much metadata as you know about your data in the read group fields provided. For more information about all the possible fields in the @RG tag, take a look at the SAM specification.

Action

Compose the read group identifier in the following format:

```
@RG\tID:group1\tSM:sample1\tPL:illumina\tLB:lib1\tPU:unit1
```

where the `\t` stands for the tab character.

---

2. Generate a SAM file containing aligned reads

Action

Run the following BWA command:

In this command, replace read group info by the read group identifier composed in the previous step.

```
bwa mem -M -R â€™<read group info>â€™ -p reference.fa raw_reads.fq > aligned_reads.sam
```

replacing the `<read group info>` bit with the read group identifier you composed at the previous step.

The `-M` flag causes BWA to mark shorter split hits as secondary (essential for Picard compatibility).

Expected Result

This creates a file called `aligned_reads.sam` containing the aligned reads from all input files, combined, annotated and aligned to the same reference.

Note that here we are using a command that is specific for pair ended data in an interleaved fastq file, which is what we are providing to you as a tutorial file. To map other types of datasets (e.g. single-ended or pair-ended in forward/reverse read files) you will need to adapt the command accordingly. Please see the BWA documentation for exact usage and more options for these commands.

---

3. Convert to BAM, sort and mark duplicates

These initial pre-processing operations format the data to suit the requirements of the GATK tools.

Action

Run the following Picard command to sort the SAM file and convert it to BAM:

```
java -jar SortSam.jar \
    INPUT=aligned_reads.sam \
    OUTPUT=sorted_reads.bam \
    SORT_ORDER=coordinate
```

Expected Results

This creates a file called `sorted_reads.bam` containing the aligned reads sorted by coordinate.

Action

Run the following Picard command to mark duplicates:

```
java -jar MarkDuplicates.jar \
    INPUT=sorted_reads.bam \
    OUTPUT=dedup_reads.bam
```

Expected Result

This creates a sorted BAM file called `dedup_reads.bam` with the same content as the input file, except that any duplicate reads are marked as such.

Action

Run the following Picard command to index the BAM file:

```
java -jar BuildBamIndex.jar \
    INPUT=dedup_reads.bam
```

Expected Result

This creates an index file for the BAM file called `dedup_reads.bai`.

(howto) Perform local realignment around indels #2800

Last updated on 2013-07-10 16:55:20

Objective

Perform local realignment around indels to correct mapping-related artifacts.

Prerequisites

- TBD

Steps

- Create a target list of intervals to be realigned

- Perform realignment of the target intervals

1. Create a target list of intervals to be realigned

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T RealignerTargetCreator \
     -R reference.fa \
     -I dedup_reads.bam \
     -L 20 \
     -known gold_indels.vcf \
     -o target_intervals.list
```

Expected Result

This creates a file called `target_intervals.list` containing the list of intervals that the program identified as needing realignment within our target, chromosome 20.

The list of known indel sites (`gold_indels.vcf`) are used as targets for realignment. Only use it if there is such a list for your organism.

---

2. Perform realignment of the target intervals

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T IndelRealigner \
     -R reference.fa \
     -I dedup_reads.bam \
     -targetIntervals target_intervals.list \
     -known gold_indels.vcf \
     -o realigned_reads.bam
```

Expected Result

This creates a file called `realigned_reads.bam` containing all the original reads, but with better local alignments in the regions that were realigned.

Note that here, we didnâ€™t include the `-L 20` argument. It's not necessary since the program will only run on the target intervals we are providing.

(howto) Prepare a reference for use with BWA and GATK #2798

Last updated on 2013-07-03 00:56:06

Objective

Prepare a reference sequence so that it is suitable for use with BWA and GATK.

Prerequisites

- Installed BWA
- Installed SAMTools
- Installed Picard

Steps

- Generate the BWA index
- Generate the Fasta file index
- Generate the sequence dictionary

---

1. Generate the BWA index

Action

Run the following BWA command:

```
bwa index -a bwtsw reference.fa
```

where `-a bwtsw` specifies that we want to use the indexing algorithm that is capable of handling the whole human genome.

Expected Result

This creates a collection of files used by BWA to perform the alignment.

---

2. Generate the fasta file index

Action

Run the following SAMtools command:

```
samtools faidx reference.fa
```

Expected Result

This creates a file called `reference.fa.fai`, with one record per line for each of the contigs in the FASTA reference file. Each record is composed of the contig name, size, location, basesPerLine and bytesPerLine.

---

3. Generate the sequence dictionary

Action

Run the following Picard command:

```
java -jar CreateSequenceDictionary.jar \
    REFERENCE=reference.fa \
    OUTPUT=reference.dict
```

Expected Result

This creates a file called `reference.dict` formatted like a SAM header, describing the contents of your reference FASTA file.

(howto) Recalibrate base quality scores = run BQSR #2801

Last updated on 2013-07-17 22:27:07

Objective

Recalibrate base quality scores in order to correct sequencing errors and other experimental artifacts.

Prerequisites

- TBD

Steps

- Analyze patterns of covariation in the sequence dataset
- Do a second pass to analyze covariation remaining after recalibration
- Generate before/after plots
- Apply the recalibration to your sequence data

---

1. Analyze patterns of covariation in the sequence dataset

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T BaseRecalibrator \
     -R reference.fa \
     -I realigned_reads.bam \
     -L 20 \
     -knownSites dbsnp.vcf \
     -knownSites gold_indels.vcf \
     -o recal_data.table
```

Expected Result

This creates a GATKReport file called `recal_data.grp` containing several tables. These tables contain the covariation data that will be used in a later step to recalibrate the base qualities of your sequence data.

It is imperative that you provide the program with a set of known sites, otherwise it will refuse to run. The known sites are used to build the covariation model and estimate empirical base qualities. For details on what to do if there are no known sites available for your organism of study, please see the online GATK documentation.

---

2. Do a second pass to analyze covariation remaining after recalibration

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T BaseRecalibrator \
     -R reference.fa \
     -I realigned_reads.bam \
     -L 20 \
     -knownSites dbsnp.vcf \
     -knownSites gold_indels.vcf \
     -BQSR recal_data.table \
     -o post_recal_data.table
```

Expected Result

This creates another GATKReport file, which we will use in the next step to generate plots. Note the use of the `-BQSR` flag, which tells the GATK engine to perform on-the-fly recalibration based on the first recalibration data table.

---

3. Generate before/after plots

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T AnalyzeCovariates \
     -R reference.fa \
     -L 20 \
     -before recal_data.table \
     -after post_recal_data.table \
     -plots recalibration_plots.pdf
```

Expected Result

This generates a document called `recalibration_plots.pdf` containing plots that show how the reported base qualities match up to the empirical qualities calculated by the BaseRecalibrator. Comparing the **before** and **after** plots allows you to check the effect of the base recalibration process before you actually apply the recalibration to your sequence data. For details on how to interpret the base recalibration plots, please see the online GATK documentation.

---

4. Apply the recalibration to your sequence data

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T PrintReads \
     -R reference.fa \
     -I realigned_reads.bam \
     -L 20 \
     -BQSR recal_data.table \
     -o recal_reads.bam
```

Expected Result

This creates a file called `recal_reads.bam` containing all the original reads, but now with exquisitely accurate base substitution, insertion and deletion quality scores. By default, the original quality scores are discarded in order to keep the file size down. However, you have the option to retain them by adding the flag `â€“emit_original_quals` to the PrintReads command, in which case the original qualities will also be written in the file, tagged `OQ`.

Notice how this step uses a very simple tool, PrintReads, to apply the recalibration. Whatâ€™s happening here is that we are loading in the original sequence data, having the GATK engine recalibrate the base qualities

on-the-fly thanks to the `-BQSR` flag (as explained earlier), and just using PrintReads to write out the resulting data to the new file.

(howto) Recalibrate variant quality scores = run VQSR #2805

Last updated on 2013-10-04 14:41:01

Objective

Recalibrate variant quality scores and produce a callset filtered for the desired levels of sensitivity and specificity.

Prerequisites

- TBD

Steps

- Prepare recalibration parameters for SNPs
 a. Specify which call sets the program should use as resources to build the recalibration model
 b. Specify which annotations the program should use to evaluate the likelihood of Indels being real
 c. Specify the desired truth sensitivity threshold values that the program should use to generate tranches
 d. Determine additional model parameters

- Build the SNP recalibration model

- Apply the desired level of recalibration to the SNPs in the call set

- Prepare recalibration parameters for Indels a. Specify which call sets the program should use as resources to build the recalibration model b. Specify which annotations the program should use to evaluate the likelihood of Indels being real c. Specify the desired truth sensitivity threshold values that the program should use to generate tranches d. Determine additional model parameters

- Build the Indel recalibration model

- Apply the desired level of recalibration to the Indels in the call set

1. Prepare recalibration parameters for SNPs

a. Specify which call sets the program should use as resources to build the recalibration model

For each training set, we use key-value tags to qualify whether the set contains known sites, training sites, and/or truth sites. We also use a tag to specify the prior likelihood that those sites are true (using the Phred scale).

- True sites training resource: HapMap

This resource is a SNP call set that has been validated to a very high degree of confidence. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). We will also use these sites later on to choose a threshold for filtering

variants based on sensitivity to truth sites. The prior likelihood we assign to these variants is Q15 (96.84%).

- True sites training resource: Omni

This resource is a set of polymorphic SNP sites produced by the Omni genotyping array. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q12 (93.69%).

- Non-true sites training resource: 1000G

This resource is a set of high-confidence SNP sites produced by the 1000 Genomes Project. The program will consider that the variants in this resource may contain true variants as well as false positives (truth=false), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q10 (%).

- Known sites resource, not used in training: dbSNP

This resource is a SNP call set that has not been validated to a high degree of confidence (truth=false). The program will not use the variants in this resource to train the recalibration model (training=false). However, the program will use these to stratify output metrics such as Ti/Tv ratio by whether variants are present in dbsnp or not (known=true). The prior likelihood we assign to these variants is Q2 (36.90%).

The default prior likelihood assigned to all other variants is Q2 (36.90%). This low value reflects the fact that the philosophy of the GATK callers is to produce a large, highly sensitive callset that needs to be heavily refined through additional filtering.

b. Specify which annotations the program should use to evaluate the likelihood of Indels being real

These annotations are included in the information generated for each variant call by the caller. If an annotation is missing (typically because it was omitted from the calling command) it can be added using the VariantAnnotator tool.

- Coverage (DP)

Total (unfiltered) depth of coverage.

- QualByDepth (QD)

Variant confidence (from the QUAL field) / unfiltered depth of non-reference samples.

- FisherStrand (FS)

Phred-scaled p-value using Fisher's Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- MappingQualityRankSumTest (MQRankSum)

The u-based z-approximation from the Mann-Whitney Rank Sum Test for mapping qualities (reads with ref

bases vs. those with the alternate allele). Note that the mapping quality rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

- ReadPosRankSumTest (ReadPosRankSum)

The u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

c. Specify the desired truth sensitivity threshold values that the program should use to generate tranches

- First tranche threshold 100.0
- Second tranche threshold 99.9
- Third tranche threshold 99.0
- Fourth tranche threshold 90.0

Tranches are essentially slices of variants, ranked by VQSLOD, bounded by the threshold values specified in this step. The threshold values themselves refer to the sensitivity we can obtain when we apply them to the call sets that the program uses to train the model. The idea is that the lowest tranche is highly specific but less sensitive (there are very few false positives but potentially many false negatives, i.e. missing calls), and each subsequent tranche in turn introduces additional true positive calls along with a growing number of false positive calls. This allows us to filter variants based on how sensitive we want the call set to be, rather than applying hard filters and then only evaluating how sensitive the call set is using post hoc methods.

d. Determine additional model parameters

- Number of bad variants (`--numBadVariants/-numBad`) 1000

This is the number of the worst scoring variants to use when building the model of bad variants.

---

2. Build the SNP recalibration model

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T VariantRecalibrator \
     -R reference.fa \
     -input raw_variants.vcf \
     -resource:hapmap,known=false,training=true,truth=true,prior=15.0 hapmap.vcf \
     -resource:omni,known=false,training=true,truth=false,prior=12.0 omni.vcf \
     -resource:1000G,known=false,training=true,truth=false,prior=10.0 1000G.vcf \
```

```
     -resource:dbsnp,known=true,training=false,truth=false,prior=2.0 dbsnp.vcf \
     -an DP \
     -an QD \
     -an FS \
     -an MQRankSum \
     -an ReadPosRankSum \
     -mode SNP \
     -tranche 100.0 -tranche 99.9 -tranche 99.0 -tranche 90.0 \
     -numBad 1000 \
     -recalFile recalibrate_SNP.recal \
     -tranchesFile recalibrate_SNP.tranches \
     -rscriptFile recalibrate_SNP_plots.R
```

Expected Result

This creates several files. The most important file is the recalibration report, called `recalibrate_SNP.recal`, which contains the recalibration data. This is what the program will use in the next step to generate a VCF file in which the variants are annotated with their recalibrated quality scores. There is also a file called `recalibrate_SNP.tranches`, which contains the quality score thresholds corresponding to the tranches specified in the original command. Finally, if your installation of R and the other required libraries was done correctly, you will also find some PDF files containing plots. These plots illustrated the distribution of variants according to certain dimensions of the model.

For detailed instructions on how to interpret these plots, please refer to the online GATK documentation.

---

3. Apply the desired level of recalibration to the SNPs in the call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T ApplyRecalibration \
     -R reference.fa \
     -input raw_variants.vcf \
     -mode SNP \
     --ts_filter_level 99.0 \
     -recalFile recalibrate_SNP.recal \
     -tranchesFile recalibrate_SNP.tranches \
     -o recalibrated_snps_raw_indels.vcf
```

Expected Result

This creates a new VCF file, called `recalibrated_snps_raw_indels.vcf`, which contains all the original variants from the original `raw_variants.vcf` file, but now the SNPs are annotated with their recalibrated

quality scores (VQSLOD) and either PASS or FILTER depending on whether or not they are included in the selected tranche.

Here we are taking the second lowest of the tranches specified in the original recalibration command. This means that we are applying to our data set the level of sensitivity that would allow us to retrieve 99% of true variants from the truth training sets of HapMap and Omni SNPs. If we wanted to be more specific (and therefore have less risk of including false positives, at the risk of missing real sites) we could take the very lowest tranche, which would only retrieve 90% of the truth training sites. If we wanted to be more sensitive (and therefore less specific, at the risk of including more false positives) we could take the higher tranches. In our Best Practices documentation, we recommend taking the second highest tranche (99.9%) which provides the highest sensitivity you can get while still being acceptably specific.

---

4. Prepare recalibration parameters for Indels

a. Specify which call sets the program should use as resources to build the recalibration model

For each training set, we use key-value tags to qualify whether the set contains known sites, training sites, and/or truth sites. We also use a tag to specify the prior likelihood that those sites are true (using the Phred scale).

- Known and true sites training resource: Mills

This resource is an Indel call set that has been validated to a high degree of confidence. The program will consider that the variants in this resource are representative of true sites (truth=true), and will use them to train the recalibration model (training=true). The prior likelihood we assign to these variants is Q12 (93.69%).

The default prior likelihood assigned to all other variants is Q2 (36.90%). This low value reflects the fact that the philosophy of the GATK callers is to produce a large, highly sensitive callset that needs to be heavily refined through additional filtering.

b. Specify which annotations the program should use to evaluate the likelihood of Indels being real

These annotations are included in the information generated for each variant call by the caller. If an annotation is missing (typically because it was omitted from the calling command) it can be added using the VariantAnnotator tool.

- Coverage (DP)

Total (unfiltered) depth of coverage.

- FisherStrand (FS)

Phred-scaled p-value using Fisher's Exact Test to detect strand bias (the variation being seen on only the forward or only the reverse strand) in the reads. More bias is indicative of false positive calls.

- MappingQualityRankSumTest (MQRankSum)

The u-based z-approximation from the Mann-Whitney Rank Sum Test for mapping qualities (reads with ref bases vs. those with the alternate allele). Note that the mapping quality rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

- ReadPosRankSumTest (ReadPosRankSum)

The u-based z-approximation from the Mann-Whitney Rank Sum Test for the distance from the end of the read for reads with the alternate allele. If the alternate allele is only seen near the ends of reads, this is indicative of error. Note that the read position rank sum test can not be calculated for sites without a mixture of reads showing both the reference and alternate alleles.

c. Specify the desired truth sensitivity threshold values that the program should use to generate tranches

- First tranche threshold 100.0
- Second tranche threshold 99.9
- Third tranche threshold 99.0
- Fourth tranche threshold 90.0

Tranches are essentially slices of variants, ranked by VQSLOD, bounded by the threshold values specified in this step. The threshold values themselves refer to the sensitivity we can obtain when we apply them to the call sets that the program uses to train the model. The idea is that the lowest tranche is highly specific but less sensitive (there are very few false positives but potentially many false negatives, i.e. missing calls), and each subsequent tranche in turn introduces additional true positive calls along with a growing number of false positive calls. This allows us to filter variants based on how sensitive we want the call set to be, rather than applying hard filters and then only evaluating how sensitive the call set is using post hoc methods.

d. Determine additional model parameters

- Number of bad variants (`--numBadVariants/-numBad`) 1000

This is the number of the worst scoring variants that the program should use when building the model of bad variants.

- Maximum number of Gaussians (`-maxGaussians`) 4

This is the maximum number of Gaussians (i.e. clusters of variants that have similar properties) that the program should try to identify when it runs the variational Bayes algorithm that underlies the machine learning method. In essence, this limits the number of different â€•profilesâ€• of variants that the program will try to identify. This number should only be increased for datasets that include very many variants.

5. Build the Indel recalibration model

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T VariantRecalibrator \
     -R reference.fa \
     -input recalibrated_snps_raw_indels.vcf \
     -resource:mills,known=true,training=true,truth=true,prior=12.0 mills.vcf \
     -an DP \
     -an FS \
     -an MQRankSum \
     -an ReadPosRankSum \
     -mode INDEL \
     -tranche 100.0 -tranche 99.9 -tranche 99.0 -tranche 90.0 \
     -numBad 1000 \
     --maxGaussians 4 \
     -recalFile recalibrate_INDEL.recal \
     -tranchesFile recalibrate_INDEL.tranches \
     -rscriptFile recalibrate_INDEL_plots.R
```

Expected Result

This creates several files. The most important file is the recalibration report, called `recalibrate_INDEL.recal`, which contains the recalibration data. This is what the program will use in the next step to generate a VCF file in which the variants are annotated with their recalibrated quality scores. There is also a file called `recalibrate_INDEL.tranches`, which contains the quality score thresholds corresponding to the tranches specified in the original command. Finally, if your installation of R and the other required libraries was done correctly, you will also find some PDF files containing plots. These plots illustrated the distribution of variants according to certain dimensions of the model.

For detailed instructions on how to interpret these plots, please refer to the online GATK documentation.

---

6. Apply the desired level of recalibration to the Indels in the call set

Action

Run the following GATK command:

```
java -jar GenomeAnalysisTK.jar \
     -T ApplyRecalibration \
     -R reference.fa \
     -input recalibrated_snps_raw_indels.vcf \
     -mode INDEL \
```

```
--ts_filter_level 99.0 \
-recalFile recalibrate_INDEL.recal \
-tranchesFile recalibrate_INDEL.tranches \
-o recalibrated_variants.vcf
```

Expected Result

This creates a new VCF file, called `recalibrated_variants.vcf`, which contains all the original variants from the original `recalibrated_snps_raw_indels.vcf` file, but now the Indels are also annotated with their recalibrated quality scores (VQSLOD) and either PASS or `FILTER` depending on whether or not they are included in the selected tranche.

Here we are taking the second lowest of the tranches specified in the original recalibration command. This means that we are applying to our data set the level of sensitivity that would allow us to retrieve 99% of true variants from the truth training sets of HapMap and Omni SNPs. If we wanted to be more specific (and therefore have less risk of including false positives, at the risk of missing real sites) we could take the very lowest tranche, which would only retrieve 90% of the truth training sites. If we wanted to be more sensitive (and therefore less specific, at the risk of including more false positives) we could take the higher tranches. In our Best Practices documentation, we recommend taking the second highest tranche (99.9%) which provides the highest sensitivity you can get while still being acceptably specific.

(howto) Revert a BAM file to FastQ format #2908

Last updated on 2013-07-31 15:57:23

Objective

Revert a BAM file back to FastQ. This comes in handy when you receive data that has been processed but not according to GATK Best Practices, and you want to reset and reprocess it properly.

Prerequisites

- Installed HTSlib

Steps

- Shuffle the reads in the bam file
- Revert the BAM file to FastQ format
- Compress the FastQ file
- Note for advanced users

1. Shuffle the reads in the bam file

Action

Shuffle the reads in the bam file so they are not in a biased order before alignment by running the following HTSlib command:

```
htscmd bamshuf -uOn 128 aln_reads.bam tmp > shuffled_reads.bam
```

Expected Result

This creates a new BAM file containing the original reads, which still retain their mapping information, but now they are no longer sorted.

The aligner uses blocks of paired reads to estimate the insert size. If you donâ€™t shuffle your original bam, the blocks of insert size will not be randomly distributed across the genome, rather they will all come from the same region, biasing the insert size calculation. This is a very important step which is unfortunately often overlooked.

---

2. Revert the BAM file to FastQ

Action

Revert the BAM file to FastQ format by running the following HTSlib command:

```
htscmd bam2fq -a shuffled_reads.bam > interleaved_reads.fq
```

Expected Result

This creates an interleaved FastQ file called `interleaved_reads.fq` containing the now-unmapped paired reads.

Interleaved simply means that for each pair of reads in your paired-end data set, both the forward and the reverse reads are in the same file, as opposed to having them in separate files.

---

3. Compress the FastQ file

Action

Compress the FastQ file to reduce its size using the gzip utility:

```
gzip interleaved_reads.fq
```

Expected Result

This creates a gzipped FastQ file called `interleaved_reads.fq.gz`. This file is ready to be used as input for the Best Practices workflow.

BWA handles gzipped fastq files natively, so you donâ€™t need to unzip the file to use it later on.

4. Note for advanced users

If youâ€™re feeling adventurous, you can do all of the above with this beautiful one-liner, which will save you a heap of time that the program would otherwise spend performing I/O (loading in and writing out data to/from disk):

```
htscmd bamshuf -uOn 128 aln_reads.bam tmp | htscmd bam2fq -a - | gzip >
interleaved_reads.fq.gz
```

(howto) Run Queue for the first time #1288

Last updated on 2013-07-08 20:13:46

Objective

Run a basic analysis command on example data, parallelized with Queue.

Prerequisites

- Successfully completed "How to test your Queue installation" and "How to run GATK for the first time"
- GATK resource bundle downloaded

Steps

- Set up a dry run of Queue
- Run the analysis for real
- Running on a computing farm

1. Set up a dry run of Queue

One very cool feature of Queue is that you can test your script by doing a "dry run". That means Queue will prepare the analysis and build the scatter commands, but not actually run them. This makes it easier to check the sanity of your script and command.

Here we're going to set up a dry run of a CountReads analysis. You should be familiar with the CountReads walker and the example files from the bundles, as used in the basic "GATK for the first time" tutorial. In addition, we're going to use the example QScript called `ExampleCountReads.scala` provided in the Queue package download.

Action

Type the following command:

```
java -Djava.io.tmpdir=tmp -jar Queue.jar -S ExampleCountReads.scala -R exampleFASTA.fasta -I exampleBAM.bam
```

where `-S ExampleCountReads.scala` specifies which QScript we want to run, `-R exampleFASTA.fasta` specifies the reference sequence, and `-I exampleBAM.bam` specifies the file of aligned reads we want to analyze.

Expected Result

After a few seconds you should see output that looks nearly identical to this:

```
INFO  00:30:45,527 QScriptManager - Compiling 1 QScript
INFO  00:30:52,869 QScriptManager - Compilation complete
INFO  00:30:53,284 HelpFormatter -
--------------------------------------------------------------------------------
INFO  00:30:53,284 HelpFormatter - Queue v2.0-36-gf5c1c1a, Compiled 2012/08/08 20:18:21
INFO  00:30:53,284 HelpFormatter - Copyright (c) 2012 The Broad Institute
INFO  00:30:53,284 HelpFormatter - Fro support and documentation go to
http://www.broadinstitute.org/gatk
INFO  00:30:53,285 HelpFormatter - Program Args: -S ExampleCountReads.scala -R
exampleFASTA.fasta -I exampleBAM.bam
INFO  00:30:53,285 HelpFormatter - Date/Time: 2012/08/09 00:30:53
INFO  00:30:53,285 HelpFormatter -
--------------------------------------------------------------------------------
INFO  00:30:53,285 HelpFormatter -
--------------------------------------------------------------------------------
INFO  00:30:53,290 QCommandLine - Scripting ExampleCountReads
INFO  00:30:53,364 QCommandLine - Added 1 functions
INFO  00:30:53,364 QGraph - Generating graph.
INFO  00:30:53,388 QGraph - -------
INFO  00:30:53,402 QGraph - Pending:  'java'  '-Xmx1024m'
'-Djava.io.tmpdir=/Users/vdauwera/sandbox/Q2/resources/tmp'  '-cp'
'/Users/vdauwera/sandbox/Q2/Queue.jar'  'org.broadinstitute.sting.gatk.CommandLineGATK'
'-T' 'CountReads'  '-I' '/Users/vdauwera/sandbox/Q2/resources/exampleBAM.bam'  '-R'
'/Users/vdauwera/sandbox/Q2/resources/exampleFASTA.fasta'
INFO  00:30:53,403 QGraph - Log:
/Users/vdauwera/sandbox/Q2/resources/ExampleCountReads-1.out
INFO  00:30:53,403 QGraph - Dry run completed successfully!
INFO  00:30:53,404 QGraph - Re-run with "-run" to execute the functions.
INFO  00:30:53,409 QCommandLine - Script completed successfully with 1 total jobs
INFO  00:30:53,410 QCommandLine - Writing JobLogging GATKReport to file
/Users/vdauwera/sandbox/Q2/resources/ExampleCountReads.jobreport.txt
```

If you don't see this, check your spelling (GATK commands are case-sensitive), check that the files are in your working directory, and if necessary, re-check that the GATK and Queue are properly installed.

If you do see this output, congratulations! You just successfully ran you first Queue dry run!

---

2. Run the analysis for real

Once you have verified that the Queue functions have been generated successfully, you can execute the pipeline by appending `-run` to the command line.

Action

Instead of this command, which we used earlier:

```
java -Djava.io.tmpdir=tmp -jar Queue.jar -S ExampleCountReads.scala -R exampleFASTA.fasta -I exampleBAM.bam
```

this time you type this:

```
java -Djava.io.tmpdir=tmp -jar Queue.jar -S ExampleCountReads.scala -R exampleFASTA.fasta -I exampleBAM.bam -run
```

See the difference?

Result

You should see output that looks nearly identical to this:

```
INFO  00:56:33,688 QScriptManager - Compiling 1 QScript
INFO  00:56:39,327 QScriptManager - Compilation complete
INFO  00:56:39,487 HelpFormatter -
---------------------------------------------------------------------
INFO  00:56:39,487 HelpFormatter - Queue v2.0-36-gf5c1c1a, Compiled 2012/08/08 20:18:21
INFO  00:56:39,488 HelpFormatter - Copyright (c) 2012 The Broad Institute
INFO  00:56:39,488 HelpFormatter - Fro support and documentation go to
http://www.broadinstitute.org/gatk
INFO  00:56:39,489 HelpFormatter - Program Args: -S ExampleCountReads.scala -R
exampleFASTA.fasta -I exampleBAM.bam -run
INFO  00:56:39,490 HelpFormatter - Date/Time: 2012/08/09 00:56:39
INFO  00:56:39,490 HelpFormatter -
---------------------------------------------------------------------
INFO  00:56:39,491 HelpFormatter -
---------------------------------------------------------------------
INFO  00:56:39,498 QCommandLine - Scripting ExampleCountReads
INFO  00:56:39,569 QCommandLine - Added 1 functions
```

```
INFO  00:56:39,569 QGraph - Generating graph.
INFO  00:56:39,589 QGraph - Running jobs.
INFO  00:56:39,623 FunctionEdge - Starting:  'java'  '-Xmx1024m'
'-Djava.io.tmpdir=/Users/vdauwera/sandbox/Q2/resources/tmp'  '-cp'
'/Users/vdauwera/sandbox/Q2/Queue.jar'  'org.broadinstitute.sting.gatk.CommandLineGATK'
'-T' 'CountReads'  '-I' '/Users/vdauwera/sandbox/Q2/resources/exampleBAM.bam'  '-R'
'/Users/vdauwera/sandbox/Q2/resources/exampleFASTA.fasta'
INFO  00:56:39,623 FunctionEdge - Output written to
/Users/GG/codespace/GATK/Q2/resources/ExampleCountReads-1.out
INFO  00:56:50,301 QGraph - 0 Pend, 1 Run, 0 Fail, 0 Done
INFO  00:57:09,827 FunctionEdge - Done:  'java'  '-Xmx1024m'
'-Djava.io.tmpdir=/Users/vdauwera/sandbox/Q2/resources/tmp'  '-cp'
'/Users/vdauwera/sandbox/Q2/resources/Queue.jar'
'org.broadinstitute.sting.gatk.CommandLineGATK'  '-T' 'CountReads'  '-I'
'/Users/vdauwera/sandbox/Q2/resources/exampleBAM.bam'  '-R'
'/Users/vdauwera/sandbox/Q2/resources/exampleFASTA.fasta'
INFO  00:57:09,828 QGraph - 0 Pend, 0 Run, 0 Fail, 1 Done
INFO  00:57:09,835 QCommandLine - Script completed successfully with 1 total jobs
INFO  00:57:09,835 QCommandLine - Writing JobLogging GATKReport to file
/Users/vdauwera/sandbox/Q2/resources/ExampleCountReads.jobreport.txt
INFO  00:57:10,107 QCommandLine - Plotting JobLogging GATKReport to file
/Users/vdauwera/sandbox/Q2/resources/ExampleCountReads.jobreport.pdf
WARN  00:57:18,597 RScriptExecutor - RScript exited with 1. Run with -l DEBUG for more info.
```

Great! It works!

The results of the traversal will be written to a file in the current directory. The name of the file will be printed in the output, ExampleCountReads.out in this example.

If for some reason the run was interrupted, in most cases you can resume by just launching the command. Queue will pick up where it left off without redoing the parts that ran successfully.

---

3. Running on a computing farm

Run with `-bsub` to run on LSF, or for early Grid Engine support see Queue with Grid Engine.

See also QFunction and Command Line Options for more info on Queue options.

(howto) Run the GATK for the first time #1209

Last updated on 2013-08-27 18:55:56

NOTICE:

This tutorial is slightly out of date so the output is a little different. We'll update this soon, but in the meantime, don't freak out if you get a result that reads something like

```
INFO 18:32:38,826 CountReads - CountReads counted 33 reads in the traversal
```

instead of

```
INFO  16:17:46,061 Walker - [REDUCE RESULT] Traversal result is: 33
```

You're doing the right thing and getting the right result.

And of course, in doubt, just post a comment on this article; we're here to answer your questions.

Objective

Run a basic analysis command on example data.

Prerequisites

- Successfully completed "How to test your GATK installation"
- Familiarity with "Input files for the GATK"
- GATK resource bundle downloaded

Steps

- Invoke the GATK CountReads command
- Further exercises

---

1. Invoke the GATK CountReads command

A very simple analysis that you can do with the GATK is getting a count of the reads in a BAM file. The GATK is capable of much more powerful analyses, but this is a good starting example because there are very few things that can go wrong.

So we are going to count the reads in the file `exampleBAM.bam`, which you can find in the GATK resource bundle along with its associated index (same file name with `.bai` extension), as well as the example reference `exampleFASTA.fasta` and its associated index (same file name with `.fai` extension) and dictionary (same

file name with `.dict` extension). Copy them to your working directory so that your directory contents look like this:

```
[bm4dd-56b:~/codespace/gatk/sandbox] vdauwera% ls -la
drwxr-xr-x  9 vdauwera  CHARLES\Domain Users     306 Jul 25 16:29 .
drwxr-xr-x@ 6 vdauwera  CHARLES\Domain Users     204 Jul 25 15:31 ..
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users    3635 Apr 10 07:39 exampleBAM.bam
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users     232 Apr 10 07:39 exampleBAM.bam.bai
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users     148 Apr 10 07:39 exampleFASTA.dict
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users  101673 Apr 10 07:39 exampleFASTA.fasta
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users      20 Apr 10 07:39
exampleFASTA.fasta.fai
```

Action

Type the following command:

```
java -jar <path to GenomeAnalysisTK.jar> -T CountReads -R exampleFASTA.fasta -I
exampleBAM.bam
```

where `-T CountReads` specifies which analysis tool we want to use, `-R exampleFASTA.fasta` specifies the reference sequence, and `-I exampleBAM.bam` specifies the file of aligned reads we want to analyze.

For any analysis that you want to run on a set of aligned reads, you will **always** need to use at least these three arguments:

- `-T` for the **t**ool name, which specifices the corresponding analysis
- `-R` for the **r**eference sequence file
- `-I` for the **i**nput BAM file of aligned reads

They don't have to be in that order in your command, but this way you can remember that you need them if you **TRI**...

Expected Result

After a few seconds you should see output that looks like to this:

```
INFO  16:17:45,945 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:17:45,946 HelpFormatter - The Genome Analysis Toolkit (GATK) v2.0-22-g40f97eb,
Compiled 2012/07/25 15:29:41
INFO  16:17:45,947 HelpFormatter - Copyright (c) 2010 The Broad Institute
INFO  16:17:45,947 HelpFormatter - For support and documentation go to
http://www.broadinstitute.org/gatk
INFO  16:17:45,947 HelpFormatter - Program Args: -T CountReads -R exampleFASTA.fasta -I
exampleBAM.bam
INFO  16:17:45,947 HelpFormatter - Date/Time: 2012/07/25 16:17:45
```

```
INFO  16:17:45,947 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:17:45,948 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:17:45,950 GenomeAnalysisEngine - Strictness is SILENT
INFO  16:17:45,982 SAMDataSource$SAMReaders - Initializing SAMRecords in serial
INFO  16:17:45,993 SAMDataSource$SAMReaders - Done initializing BAM readers: total time 0.01

INFO  16:17:46,060 TraversalEngine - [INITIALIZATION COMPLETE; TRAVERSAL STARTING]
INFO  16:17:46,060 TraversalEngine -        Location processed.reads  runtime per.1M.reads
completed total.runtime remaining
INFO  16:17:46,061 Walker - [REDUCE RESULT] Traversal result is: 33
INFO  16:17:46,061 TraversalEngine - Total runtime 0.00 secs, 0.00 min, 0.00 hours
INFO  16:17:46,100 TraversalEngine - 0 reads were filtered out during traversal out of 33
total (0.00%)
INFO  16:17:46,729 GATKRunReport - Uploaded run statistics report to AWS S3
```

Depending on the GATK release, you may see slightly different information output, but you know everything is running correctly if you see the line:

```
INFO  21:53:04,556 Walker - [REDUCE RESULT] Traversal result is: 33
```

somewhere in your output.

If you don't see this, check your spelling (GATK commands are case-sensitive), check that the files are in your working directory, and if necessary, re-check that the GATK is properly installed.

If you do see this output, congratulations! You just successfully ran you first GATK analysis!

Basically the output you see means that the CountReadsWalker (which you invoked with the command line option `-T CountReads`) counted 33 reads in the `exampleBAM.bam` file, which is exactly what we expect to see.

Wait, what is this walker thing?

In the GATK jargon, we call the tools walkers because the way they work is that they walk through the dataset --either along the reference sequence (LocusWalkers), or down the list of reads in the BAM file (ReadWalkers)-- collecting the requested information along the way.

---

2. Further Exercises

Now that you're rocking the read counts, you can start to expand your use of the GATK command line.

Let's say you don't care about counting reads anymore; now you want to know the number of loci (positions on the genome) that are covered by one or more reads. The name of the tool, or walker, that does this is `CountLoci`. Since the structure of the GATK command is basically always the same, you can simply switch the tool name, right?

Action

Instead of this command, which we used earlier:

```
java -jar <path to GenomeAnalysisTK.jar> -T CountReads -R exampleFASTA.fasta -I exampleBAM.bam
```

this time you type this:

```
java -jar <path to GenomeAnalysisTK.jar> -T CountLoci -R exampleFASTA.fasta -I exampleBAM.bam
```

See the difference?

Result

You should see something like this output:

```
INFO  16:18:26,183 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:18:26,185 HelpFormatter - The Genome Analysis Toolkit (GATK) v2.0-22-g40f97eb,
Compiled 2012/07/25 15:29:41
INFO  16:18:26,185 HelpFormatter - Copyright (c) 2010 The Broad Institute
INFO  16:18:26,185 HelpFormatter - For support and documentation go to
http://www.broadinstitute.org/gatk
INFO  16:18:26,186 HelpFormatter - Program Args: -T CountLoci -R exampleFASTA.fasta -I
exampleBAM.bam
INFO  16:18:26,186 HelpFormatter - Date/Time: 2012/07/25 16:18:26
INFO  16:18:26,186 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:18:26,186 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:18:26,189 GenomeAnalysisEngine - Strictness is SILENT
INFO  16:18:26,222 SAMDataSource$SAMReaders - Initializing SAMRecords in serial
INFO  16:18:26,233 SAMDataSource$SAMReaders - Done initializing BAM readers: total time 0.01

INFO  16:18:26,351 TraversalEngine - [INITIALIZATION COMPLETE; TRAVERSAL STARTING]
INFO  16:18:26,351 TraversalEngine -        Location processed.sites  runtime per.1M.sites
completed total.runtime remaining
2052
INFO  16:18:26,411 TraversalEngine - Total runtime 0.08 secs, 0.00 min, 0.00 hours
INFO  16:18:26,450 TraversalEngine - 0 reads were filtered out during traversal out of 33
total (0.00%)
```

```
INFO  16:18:27,124 GATKRunReport - Uploaded run statistics report to AWS S3
```

Great! But wait -- where's the result? Last time the result was given on this line:

```
INFO  21:53:04,556 Walker - [REDUCE RESULT] Traversal result is: 33
```

But this time there is no line that says `[REDUCE RESULT]`! Is something wrong?

Not really. The program ran just fine -- but we forgot to give it an output file name. You see, the `CountLoci` walker is set up to output the result of its calculations to a text file, unlike `CountReads`, which is perfectly happy to output its result to the terminal screen.

Action

So we repeat the command, but this time we specify an output file, like this:

```
java -jar <path to GenomeAnalysisTK.jar> -T CountLoci -R exampleFASTA.fasta -I
exampleBAM.bam -o output.txt
```

where `-o` (lowercase o, not zero) is used to specify the output.

Result

You should get essentially the same output on the terminal screen as previously (but notice the difference in the line that contains `Program Args` -- the new argument is included):

```
INFO  16:29:15,451 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:29:15,453 HelpFormatter - The Genome Analysis Toolkit (GATK) v2.0-22-g40f97eb,
Compiled 2012/07/25 15:29:41
INFO  16:29:15,453 HelpFormatter - Copyright (c) 2010 The Broad Institute
INFO  16:29:15,453 HelpFormatter - For support and documentation go to
http://www.broadinstitute.org/gatk
INFO  16:29:15,453 HelpFormatter - Program Args: -T CountLoci -R exampleFASTA.fasta -I
exampleBAM.bam -o output.txt
INFO  16:29:15,454 HelpFormatter - Date/Time: 2012/07/25 16:29:15
INFO  16:29:15,454 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:29:15,454 HelpFormatter -
--------------------------------------------------------------------------------
INFO  16:29:15,457 GenomeAnalysisEngine - Strictness is SILENT
INFO  16:29:15,488 SAMDataSource$SAMReaders - Initializing SAMRecords in serial
INFO  16:29:15,499 SAMDataSource$SAMReaders - Done initializing BAM readers: total time 0.01

INFO  16:29:15,618 TraversalEngine - [INITIALIZATION COMPLETE; TRAVERSAL STARTING]
INFO  16:29:15,618 TraversalEngine -        Location processed.sites  runtime per.1M.sites
completed total.runtime remaining
```

```
INFO  16:29:15,679 TraversalEngine - Total runtime 0.08 secs, 0.00 min, 0.00 hours
INFO  16:29:15,718 TraversalEngine - 0 reads were filtered out during traversal out of 33
total (0.00%)
INFO  16:29:16,712 GATKRunReport - Uploaded run statistics report to AWS S3
```

This time however, if we look inside the working directory, there is a newly created file there called `output.txt` .

```
[bm4dd-56b:~/codespace/gatk/sandbox] vdauwera% ls -la
drwxr-xr-x  9 vdauwera  CHARLES\Domain Users     306 Jul 25 16:29 .
drwxr-xr-x@ 6 vdauwera  CHARLES\Domain Users     204 Jul 25 15:31 ..
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users    3635 Apr 10 07:39 exampleBAM.bam
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users     232 Apr 10 07:39 exampleBAM.bam.bai
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users     148 Apr 10 07:39 exampleFASTA.dict
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users  101673 Apr 10 07:39 exampleFASTA.fasta
     -rw-r--r--@ 1 vdauwera  CHARLES\Domain Users      20 Apr 10 07:39
exampleFASTA.fasta.fai
     -rw-r--r--  1 vdauwera  CHARLES\Domain Users       5 Jul 25 16:29 output.txt
```

This file contains the result of the analysis:

```
[bm4dd-56b:~/codespace/gatk/sandbox] vdauwera% cat output.txt
2052
```

This means that there are 2052 loci in the reference sequence that are covered by at least one or more reads in the BAM file.

Discussion

Okay then, but why not show the full, correct command in the first place? Because this was a good opportunity for you to learn a few of the caveats of the GATK command system, which may save you a lot of frustration later on.

Beyond the common basic arguments that almost all GATK walkers require, most of them also have specific requirements or options that are important to how they work. You should always check what are the specific arguments that are required, recommended and/or optional for the walker you want to use before starting an analysis.

Fortunately the GATK is set up to complain (i.e. terminate with an error message) if you try to run it without specifying a required argument. For example, if you try to run this:

```
java -jar <path to GenomeAnalysisTK.jar> -T CountLoci -R exampleFASTA.fasta
```

the GATK will spit out a wall of text, including the basic usage guide that you can invoke with the `--help` option, and more importantly, the following error message:

```
##### ERROR
```

```
--------------------------------------------------------------------------------
##### ERROR A USER ERROR has occurred (version 2.0-22-g40f97eb):
##### ERROR The invalid arguments or inputs must be corrected before the GATK can proceed
##### ERROR Please do not post this error to the GATK forum
##### ERROR
##### ERROR See the documentation (rerun with -h) for this tool to view allowable
command-line arguments.
##### ERROR Visit our website and forum for extensive documentation and answers to
##### ERROR commonly asked questions http://www.broadinstitute.org/gatk
##### ERROR
##### ERROR MESSAGE: Walker requires reads but none were provided.
##### ERROR
--------------------------------------------------------------------------------
```

You see the line that says `ERROR MESSAGE: Walker requires reads but none were provided`? This tells you exactly what was wrong with your command.

So the GATK will not run if a walker does not have all the required inputs. That's a good thing! But in the case of our first attempt at running `CountLoci`, the `-o` argument is not required by the GATK to run -- it's just highly desirable if you actually want the result of the analysis!

There will be many other cases of walkers with arguments that are not strictly required, but highly desirable if you want the results to be meaningful.

So, at the risk of getting repetitive, **always read the documentation of each walker that you want to use!**

(howto) Test your GATK installation #1200

Last updated on 2013-07-08 19:54:00

Objective

Test that the GATK is correctly installed, and that the supporting tools like Java are in your path.

Prerequisites

- Basic familiarity with the command-line environment
- Understand what is a PATH variable
- GATK downloaded and placed on path

Steps

- Invoke the GATK usage/help message
- Troubleshooting

1. Invoke the GATK usage/help message

The command we're going to run is a very simple command that asks the GATK to print out a list of available command-line arguments and options. It is so simple that it will ALWAYS work if your GATK package is installed correctly.

Note that this command is also helpful when you're trying to remember something like the right spelling or short name for an argument and for whatever reason you don't have access to the web-based documentation.

Action

Type the following command:

```
java -jar <path to GenomeAnalysisTK.jar> --help
```

replacing the `<path to GenomeAnalysisTK.jar>` bit with the path you have set up in your command-line environment.

Expected Result

You should see usage output similar to the following:

```
usage: java -jar GenomeAnalysisTK.jar -T <analysis_type> [-I <input_file>] [-L
        <intervals>] [-R <reference_sequence>] [-B <rodBind>] [-D <DBSNP>] [-H
        <hapmap>] [-hc <hapmap_chip>] [-o <out>] [-e <err>] [-oe <outerr>] [-A] [-M
        <maximum_reads>] [-sort <sort_on_the_fly>] [-compress <bam_compression>] [-fmq0]
[-dfrac
        <downsample_to_fraction>] [-dcov <downsample_to_coverage>] [-S
        <validation_strictness>] [-U] [-P] [-dt] [-tblw] [-nt <numthreads>] [-l
        <logging_level>] [-log <log_to_file>] [-quiet] [-debug] [-h]
     -T,--analysis_type <analysis_type>                    Type of analysis to run
     -I,--input_file <input_file>                          SAM or BAM file(s)
     -L,--intervals <intervals>                            A list of genomic intervals over
which
                                                       to operate. Can be explicitly
specified
                                                       on the command line or in a file.
     -R,--reference_sequence <reference_sequence>          Reference sequence file
     -B,--rodBind <rodBind>                                Bindings for reference-ordered
data, in
                                                       the form <name>,<type>,<file>
     -D,--DBSNP <DBSNP>                                    DBSNP file
     -H,--hapmap <hapmap>                                  Hapmap file
     -hc,--hapmap_chip <hapmap_chip>                       Hapmap chip file
     -o,--out <out>                                        An output file presented to the
walker.
                                                       Will overwrite contents if file
```

```
exists.
     -e,--err <err>                                    An error output file presented
to the
                                                       walker. Will overwrite contents if
file
                                                       exists.
     -oe,--outerr <outerr>                             A joint file for 'normal' and
error
                                                       output presented to the walker. Will
                                                       overwrite contents if file exists.

...
```

If you see this message, your GATK installation is ok. You're good to go! If you don't see this message, and instead get an error message, proceed to the next section on troubleshooting.

---

2. Troubleshooting

Let's try to figure out what's not working.

Action

First, make sure that your Java version is at least 1.7, by typing the following command:

```
java -version
```

Expected Result

You should see something similar to the following text:

```
java version "1.7.0_12"
Java(TM) SE Runtime Environment (build 1.7.0_12-b04)
Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed mode)
```

Remedial actions

If the version is less then 1.7, install the newest version of Java onto the system. If you instead see something like

```
java: Command not found
```

make sure that java is installed on your machine, and that your PATH variable contains the path to the java executables.

(howto) Test your Queue installation #1287

Last updated on 2013-06-17 21:09:33

Objective

Test that Queue is correctly installed, and that the supporting tools like Java are in your path.

Prerequisites

- Basic familiarity with the command-line environment
- Understand what is a PATH variable
- GATK installed
- Queue downloaded and placed on path

Steps

- Invoke the Queue usage/help message
- Troubleshooting

---

1. Invoke the Queue usage/help message

The command we're going to run is a very simple command that asks Queue to print out a list of available command-line arguments and options. It is so simple that it will ALWAYS work if your Queue package is installed correctly.

Note that this command is also helpful when you're trying to remember something like the right spelling or short name for an argument and for whatever reason you don't have access to the web-based documentation.

Action

Type the following command:

```
java -jar <path to Queue.jar> --help
```

replacing the `<path to Queue.jar>` bit with the path you have set up in your command-line environment.

Expected Result

You should see usage output similar to the following:

```
usage: java -jar Queue.jar -S <script> [-jobPrefix <job_name_prefix>] [-jobQueue <job_queue>
] [-jobProject <job_project>]
       [-jobSGDir <job_scatter_gather_directory>] [-memLimit <default_memory_limit>]
[-runDir <run_directory>] [-tempDir
       <temp_directory>] [-emailHost <emailSmtpHost>] [-emailPort <emailSmtpPort>]
[-emailTLS] [-emailSSL] [-emailUser
```

```
       <emailUsername>] [-emailPass <emailPassword>] [-emailPassFile <emailPasswordFile>]
[-bsub] [-run] [-dot <dot_graph>]
       [-expandedDot <expanded_dot_graph>] [-startFromScratch] [-status] [-statusFrom <
status_email_from>] [-statusTo
       <status_email_to>] [-keepIntermediates] [-retry <retry_failed>] [-l <logging_level>]
[-log <log_to_file>] [-quiet]
       [-debug] [-h]

     -S,--script <script>                                                    QScript scala
file
     -jobPrefix,--job_name_prefix <job_name_prefix>                          Default name
prefix for compute farm jobs.
     -jobQueue,--job_queue <job_queue>                                       Default queue
for compute farm jobs.
     -jobProject,--job_project <job_project>                                 Default
project for compute farm jobs.
     -jobSGDir,--job_scatter_gather_directory <job_scatter_gather_directory>   Default
directory to place scatter gather
                                                                           output for
compute farm jobs.
     -memLimit,--default_memory_limit <default_memory_limit>                 Default
memory limit for jobs, in gigabytes.
     -runDir,--run_directory <run_directory>                                 Root
directory to run functions from.
     -tempDir,--temp_directory <temp_directory>                              Temp
directory to pass to functions.
     -emailHost,--emailSmtpHost <emailSmtpHost>                              Email SMTP
host. Defaults to localhost.
     -emailPort,--emailSmtpPort <emailSmtpPort>                              Email SMTP
port. Defaults to 465 for ssl,
                                                                           otherwise 25.
     -emailTLS,--emailUseTLS                                                 Email should
use TLS. Defaults to false.
     -emailSSL,--emailUseSSL                                                 Email should
use SSL. Defaults to false.
     -emailUser,--emailUsername <emailUsername>                              Email SMTP
username. Defaults to none.
     -emailPass,--emailPassword <emailPassword>                              Email SMTP
password. Defaults to none. Not
                                                                           secure! See
emailPassFile.
     -emailPassFile,--emailPasswordFile <emailPasswordFile>                  Email SMTP
password file. Defaults to none.
     -bsub,--bsub_all_jobs                                                   Use bsub to
submit jobs
     -run,--run_scripts                                                      Run QScripts.
```

```
 Without this flag set only
                                                                                performs a dry
run.
     -dot,--dot_graph <dot_graph>                                                   Outputs the
queue graph to a .dot file.  See:

http://en.wikipedia.org/wiki/DOT_language
     -expandedDot,--expanded_dot_graph <expanded_dot_graph>                         Outputs the
queue graph of scatter gather to
                                                                                a .dot file.
Otherwise overwrites the
                                                                                dot_graph
     -startFromScratch,--start_from_scratch                                     Runs all
command line functions even if the
                                                                                outputs were
previously output successfully.
     -status,--status                                                           Get status of
jobs for the qscript
     -statusFrom,--status_email_from <status_email_from>                        Email address
to send emails from upon
                                                                                completion or on
error.
     -statusTo,--status_email_to <status_email_to>                              Email address
to send emails to upon
                                                                                completion or on
error.
     -keepIntermediates,--keep_intermediate_outputs                             After a
successful run keep the outputs of
                                                                                any Function
marked as intermediate.
     -retry,--retry_failed <retry_failed>                                       Retry the
specified number of times after a
                                                                                command fails.
Defaults to no retries.
     -l,--logging_level <logging_level>                                         Set the
minimum level of logging, i.e.
                                                                                setting INFO
get's you INFO up to FATAL,
                                                                                setting ERROR
gets you ERROR and FATAL level
                                                                                logging.
     -log,--log_to_file <log_to_file>                                           Set the
logging location
     -quiet,--quiet_output_mode                                                 Set the
logging to quiet mode, no output to
                                                                                stdout
```

```
     -debug,--debug_mode                                                    Set the
logging file string to include a lot
                                                                          of debugging
information (SLOW!)
     -h,--help                                                              Generate this
help message
```

If you see this message, your Queue installation is ok. You're good to go! If you don't see this message, and instead get an error message, proceed to the next section on troubleshooting.

---

2. Troubleshooting

Let's try to figure out what's not working.

Action

First, make sure that your Java version is at least 1.6, by typing the following command:

```
java -version
```

Expected Result

You should see something similar to the following text:

```
java version "1.6.0_12"
Java(TM) SE Runtime Environment (build 1.6.0_12-b04)
Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed mode)
```

Remedial actions

If the version is less then 1.6, install the newest version of Java onto the system. If you instead see something like

```
java: Command not found
```

make sure that java is installed on your machine, and that your PATH variable contains the path to the java executables.

On a Mac running OS X 10.5+, you may need to run /Applications/Utilities/Java Preferences.app and drag Java SE 6 to the top to make your machine run version 1.6, even if it has been installed.

Developer Zone

This section contains articles related to developing for the GATK. Topics covered include how to write new walkers and Queue scripts, as well as some deeper GATK engine information that is relevant for developers.

Accessing reads: AlignmentContext and ReadBackedPileup #1322

Last updated on 2012-10-18 15:36:32

1. Introduction

The AlignmentContext and ReadBackedPileup work together to provide the read data associated with a given locus. This section details the tools the GATK provides for working with collections of aligned reads.

2. What are read backed pileups?

Read backed pileups are objects that contain all of the reads and their offsets that "pile up" at a locus on the genome. They are the basic input data for the GATK LocusWalkers, and underlie most of the locus-based analysis tools like the recalibrator and SNP caller. Unfortunately, there are many ways to view this data, and version one grew unwieldy trying to support all of these approaches. Version two of the ReadBackedPileup presents a consistent and clean interface for working pileup data, as well as supporting the `iterable()` interface to enable the convenient `for ( PileupElement p : pileup )` for-each loop support.

3. How do I get a ReadBackedPileup and/or how do I create one?

The best way is simply to grab the pileup (the underlying representation of the locus data) from your `AlignmentContext` object in `map`:

```
public Integer map(RefMetaDataTracker tracker, ReferenceContext ref, AlignmentContext
context)
    ReadBackedPileup pileup = context.getPileup();
```

This aligns your calculations with the GATK core infrastructure, and avoids any unnecessary data copying from the engine to your walker.

If you are trying to create your own, the best constructor is:

```
public ReadBackedPileup(GenomeLoc loc, ArrayList<PileupElement> pileup )
```

requiring only a list, in order of read / offset in the pileup, of PileupElements.

From List and List

If you happen to have lists of SAMRecords and integer offsets into them you can construct a `ReadBackedPileup` this way:

```
public ReadBackedPileup(GenomeLoc loc, List<SAMRecord> reads, List<Integer> offsets )
```

4. What's the best way to use them?

Best way if you just need reads, bases and quals

```
for ( PileupElement p : pileup ) {
  System.out.printf("%c %c %d%n", p.getBase(), p.getSecondBase(), p.getQual());
  // you can get the read itself too using p.getRead()
}
```

This is the most efficient way to get data, and should be used whenever possible.

I just want a vector of bases and quals

You can use:

```
public byte[] getBases()
public byte[] getSecondaryBases()
public byte[] getQuals()
```

To get the bases and quals as a `byte[]` array, which is the underlying base representation in the SAM-JDK.

All I care about are counts of bases

Use the follow function to get counts of A, C, G, T in order:

```
public int[] getBaseCounts()
```

Which returns a `int[4]` vector with counts according to `BaseUtils.simpleBaseToBaseIndex` for each base.

Can I view just the reads for a given sample, read group, or any other arbitrary filter?

The GATK can very efficiently stratify pileups by sample, and less efficiently stratify by read group, strand, mapping quality, base quality, or any arbitrary filter function. The sample-specific functions can be called as follows:

```
pileup.getSamples();
pileup.getPileupForSample(String sampleName);
```

In addition to the rich set of filtering primitives built into the `ReadBackedPileup`, you can supply your own primitives by implmenting a PileupElementFilter:

```
public interface PileupElementFilter {
    public boolean allow(final PileupElement pileupElement);
}
```

and passing it to `ReadBackedPileup`'s generic filter function:

```
public ReadBackedPileup getFilteredPileup(PileupElementFilter filter);
```

See the `ReadBackedPileup`'s java documentation for a complete list of built-in filtering primitives.

Historical: StratifiedAlignmentContext

While `ReadBackedPileup` is the preferred mechanism for aligned reads, some walkers still use the `StratifiedAlignmentContext` to carve up selections of reads. If you find functions that you require in `StratifiedAlignmentContext` that seem to have no analog in `ReadBackedPileup`, please let us know and we'll port the required functions for you.

Adding and updating dependencies #1352

Last updated on 2013-03-25 18:23:56

Adding Third-party Dependencies

The GATK build system uses the Ivy dependency manager to make it easy for our users to add additional dependencies. Ivy can pull the latest jars and their dependencies from the Maven repository, making adding or updating a dependency as simple as adding a new line to the `ivy.xml` file.

If your tool is available in the maven repository, add a line to the `ivy.xml` file similar to the following:

```
<dependency org="junit" name="junit" rev="4.4" />
```

If you would like to add a dependency to a tool not available in the maven repository, please email gsahelp@broadinstitute.org

Updating SAM-JDK and Picard

Because we work so closely with the SAM-JDK/Picard team and are critically dependent on the code they produce, we have a special procedure for updating the SAM/Picard jars. Please use the following procedure to when updating `sam-*.jar` or `picard-*.jar`.

- Download and build the latest versions of Picard public and Picard private (restricted to Broad Institute users) from their respective svns.

- Get the latest svn versions for picard public and picard private by running the following commands:

svn info $PICARD_PUBLIC_HOME | grep "Revision" svn info $PICARD_PRIVATE_HOME | grep "Revision"

Updating the Picard public jars

- Rename the jars and xmls in `$STING_HOME/settings/repository/net.sf` to `{picard|sam}-$PICARD_PUBLIC_MAJOR_VERSION.$PICARD_PUBLIC_MINOR_VERSION.PICARD_PUBLIC_SVN_REV.{jar|xml}`

- Update the jars in `$STING_HOME/settings/repository/net.sf` with their newer equivalents in

$PICARD_PUBLIC_HOME/dist/picard_lib.

- Update the xmls in $STING_HOME/settings/repository/net.sf with the appropriate version number ($PICARD_PUBLIC_MAJOR_VERSION.$PICARD_PUBLIC_MINOR_VERSION.$PICARD_PUBLIC_SVN_REV).

Updating the Picard private jar

- Create the picard private jar with the following command:

ant clean package -Dexecutable=PicardPrivate -Dpicard.dist.dir=${PICARD_PRIVATE_HOME}/dist

- Rename picard-private-parts-*.jar in $STING_HOME/settings/repository/edu.mit.broad to picard-private-parts-$PICARD_PRIVATE_SVN_REV.jar.

- Update picard-private-parts-*.jar in $STING_HOME/settings/repository/edu.mit.broad with the picard-private-parts.jar in $STING_HOME/dist/packages/picard-private-parts.

- Update the xml in $STING_HOME/settings/repository/edu.mit.broad to reflect the new revision and publication date.

Clover coverage analysis with ant #2002

Last updated on 2013-01-31 19:09:42

Introduction

This document describes the workflow we use within GSA to do coverage analysis of the GATK codebase. It is primarily meant as an internal reference for team members, but are making it public to provide an example of how we work. There are a few mentions of internal server names etc.; please just disregard those as they will not be applicable to you.

Build the GATK, and run tests with clover

```
ant clean with.clover unittest
```

Note that you have to explicitly disable scala (due to a limitation in how it's currently integrated in build.xml). Note you can use things like -Dsingle="ReducerUnitTest" as well.

It seems that clover requires a lot of memory, so a few things are necessary:

```
setenv ANT_OPTS "-Xmx8g"
```

There's plenty of memory on gsa4, so it's not a problem to require so much memory

Getting more detailed reports

You can add the argument -Dclover.instrument.level=statement if you want line-level resolution on the report, but note this is astronomically expensive for the entire unit test suite. It's fine though if you want to run specific run tests.

Generate the report

```
> ant clover.report
Buildfile: /Users/depristo/Desktop/broadLocal/GATK/unstable/build.xml

clover.report:
[clover-html-report] Clover Version 3.1.8, built on November 13 2012 (build-876)
[clover-html-report] Loaded from:
/Users/depristo/Desktop/broadLocal/GATK/unstable/private/resources/clover/lib/clover.jar
[clover-html-report] Clover: Community License registered to Broad Institute.
[clover-html-report] Loading coverage database from:
'/Users/depristo/Desktop/broadLocal/GATK/unstable/.clover/clover3_1_8.db'
[clover-html-report] Writing HTML report to
'/Users/depristo/Desktop/broadLocal/GATK/unstable/clover_html'
[clover-html-report] Done. Processed 132 packages in 20943ms (158ms per package).
    [mkdir] Created dir: /Users/depristo/private_html/report/clover
     [copy] Copying 4545 files to /Users/depristo/private_html/report/clover

BUILD SUCCESSFUL
```

The clover files are present in a subdirectory clover_html as well as copied to your private_html/report directory. Note this can be very expensive given our large number of tests. For example, I've been waiting for the report to generate for nearly an hour on gsa4.

Doing it all at once

```
ant clean with.clover unittest clover.report
```

will clean the source, rebuild with clover engaged, run the unit tests, and generate the clover report. Note that currently unittests may be failing due to classcast and other exceptions in the clover run. We're looking into it. But you can still run clover.report after the failed run, as the db contains all of the run information, even through it failed (though failed methods won't be counted).

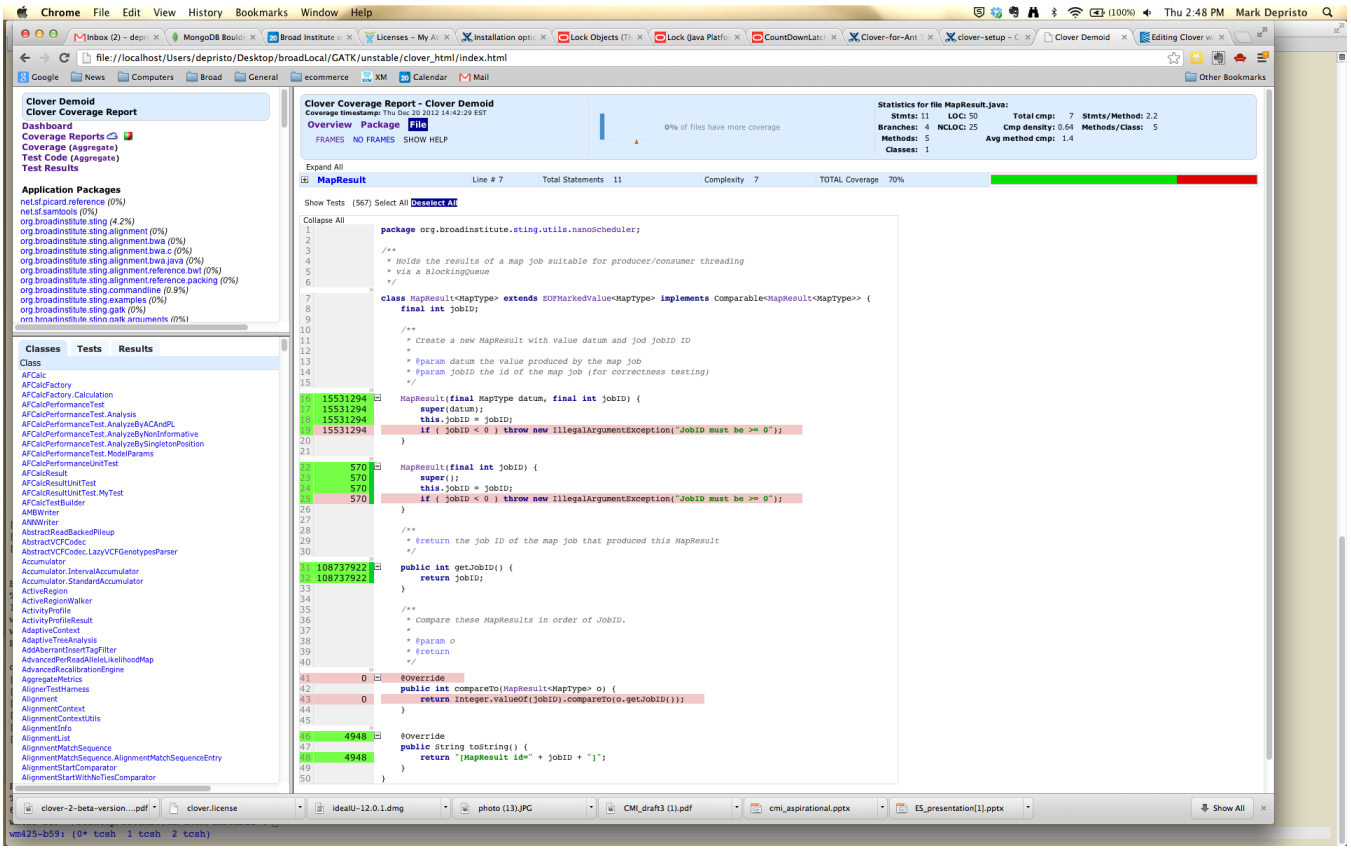
Here's a real-life example of assessing coverage in all BQSR utilities at once:

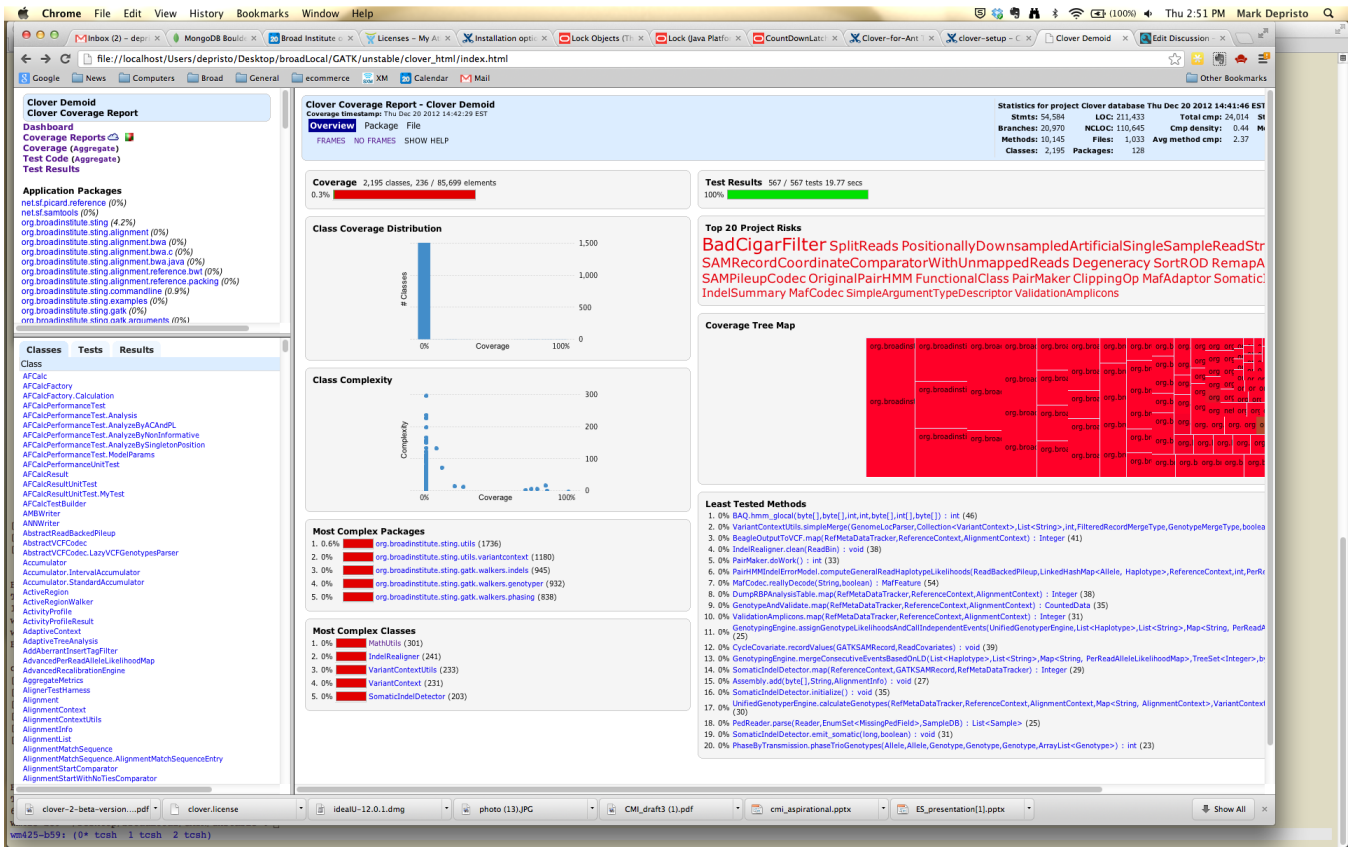
```
ant clean with.clover unittest -Dclover.instrument.level=statement
-Dsingle="recalibration/*UnitTest" clover.report
```

Current annoyance

Clover can make the tests very slow. Currently we are run in method count only mode (we don't have line number resolution (looking into fixing this). Also note that running with clover over the entire unittest set requires 32G of RAM (set automatically by ant).

This produces an HTML report that looks like the following screenshots





Using clover to make better unittests

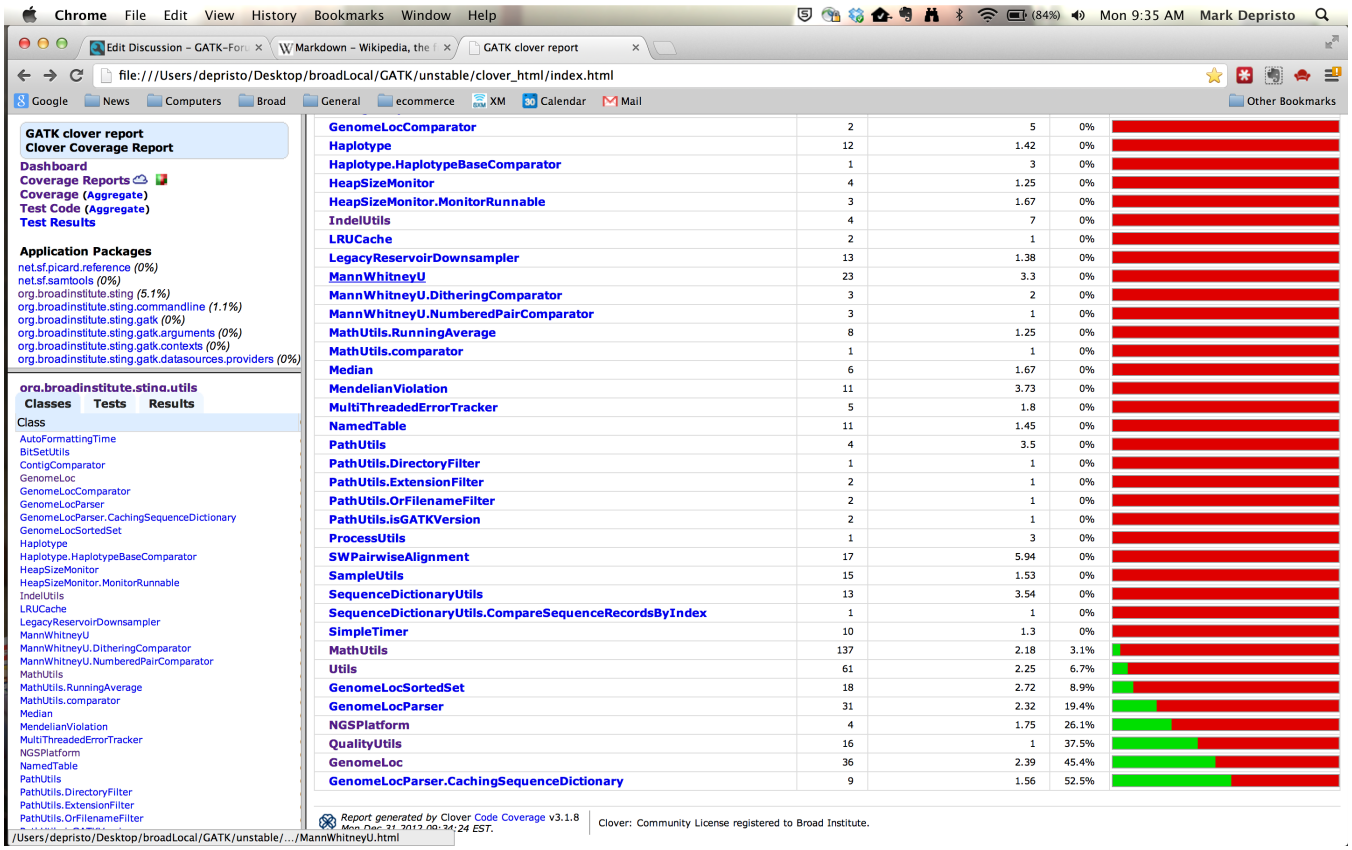
This workflow is appropriate for developing unit tests for a single package or class. The turn-around time for clover on a single package is very fast, even with statement-level coverage. The overall workflow looks like:

- run unittests with clover enabled for your package or class.
- explore clover HTML report, noting places where test coverage is lacking
- expand unit tests
- repeat until satisfied

Here's a concrete example. Right now I'm looking at the unit test coverage for GenomeLoc, one of the earliest and most important classes in the GATK. I really want good unit test coverage here. So I start by running GenomeLoc unit tests specifically:

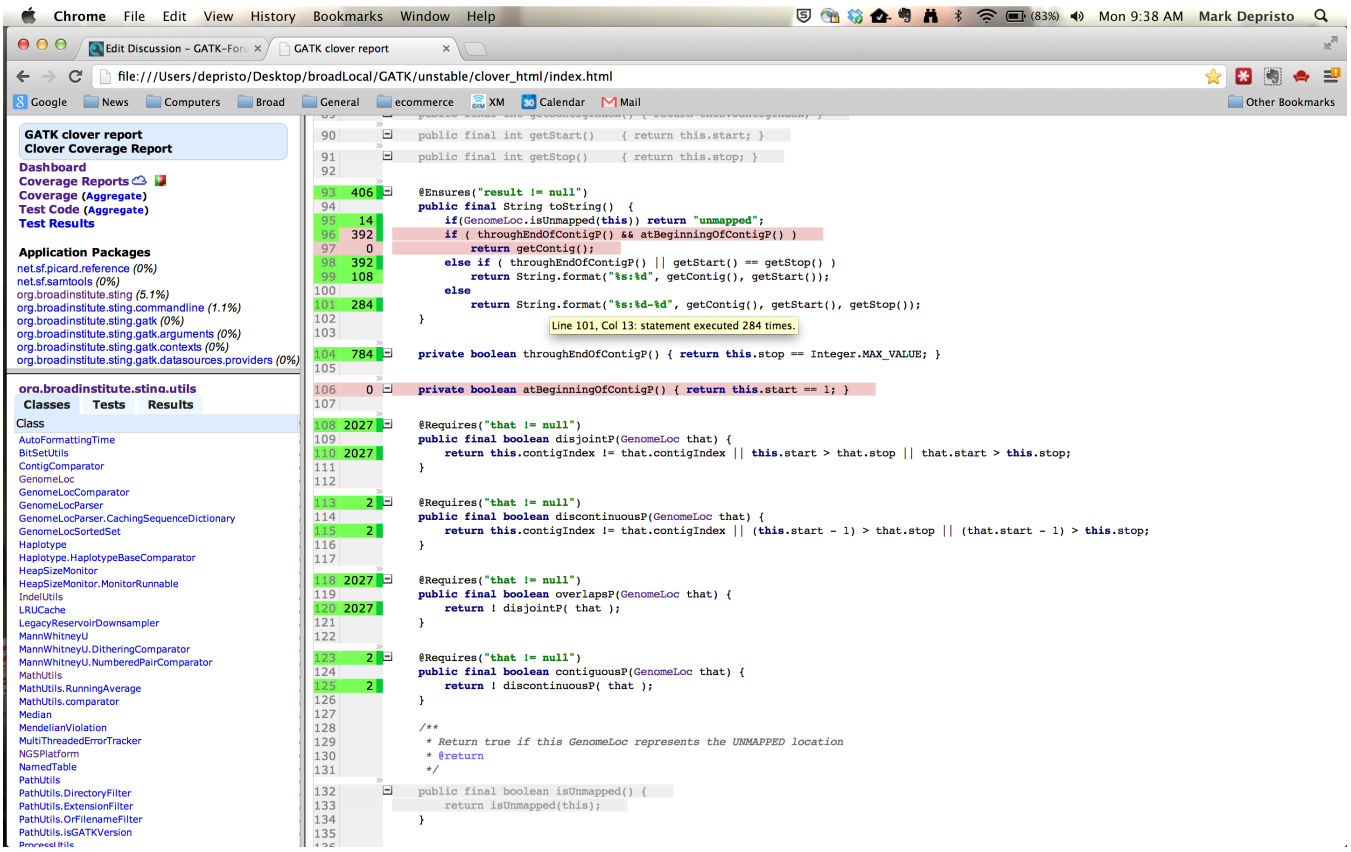
```
ant clean with.clover unittest -Dsingle="GenomeLocUnitTest"
-Dclover.instrument.level=statement clover.report
```

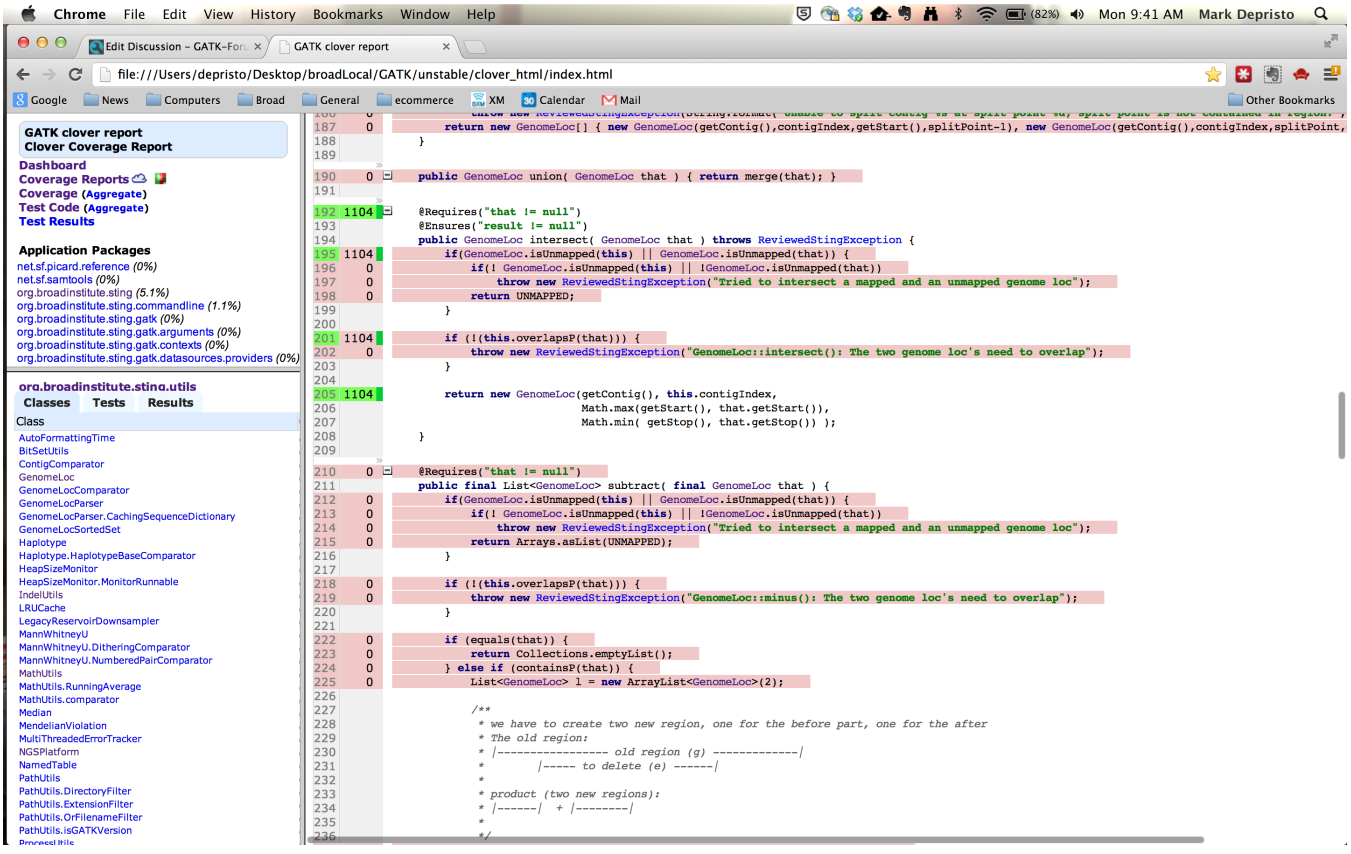
Next, I open up the clover coverage report in clover_html/index.html in my GATK directory, and landing on the Dashboard. Everything looks pretty bad, but that's because I only ran the GenomeLoc tests, and it displays the entire project coverage. I click on the "Coverage" link in the upper-left frame, and scroll down to the package where GenomeLoc lives (org.broadinstitute.sting.utils). At the bottom of this page I find my two classes, GenomeLoc and GenomeLocParser.CachingSequenceDictionary:



These have ~50% statement-level coverage each. Not ideal, really.

Let's dive into GenomeLoc itself a bit more. Clicking on the GenomeLoc link brings up to the code coverage page. Here you can see a few things very quickly.


Chrome File Edit View History Bookmarks Window Help
Mon 9:38 AM Mark Depristo
Edit Discussion - GATK-For
GATK clover report
file:///Users/depristo/Desktop/broadLocal/GATK/unstable/clover_html/index.html
Google News Computers Broad General ecommerce XM Calendar Mail Other Bookmarks
GATK clover report
Clover Coverage Report
Dashboard
Coverage Reports
Coverage (Aggregate)
Test Code (Aggregate)
Test Results
Application Packages
net.sf.picard.reference (0%)
net.sf.samtools (0%)
org.broadinstitute.sting (5.1%)
org.broadinstitute.sting.commandline (1.1%)
org.broadinstitute.sting.gatk (0%)
org.broadinstitute.sting.gatk.arguments (0%)
org.broadinstitute.sting.gatk.contexts (0%)
org.broadinstitute.sting.gatk.datasources.providers (0%)
org.broadinstitute.sting.utils
Classes Tests Results
Class
AutoFormattingTime
BitSetUtils
ContigComparator
GenomeLoc
GenomeLocComparator
GenomeLocParser
GenomeLocParser.CachingSequenceDictionary
GenomeLocSortedSet
Haplotype
Haplotype.HaplotypeBaseComparator
HeapSizeMonitor
HeapSizeMonitor.MonitorRunnable
IndelUtils
LRUCache
LegacyReservoirDownsampler
MannWhitneyU
MannWhitneyU.DitheringComparator
MannWhitneyU.NumberedPairComparator
MathUtils
MathUtils.RunningAverage
MathUtils.comparator
Median
MendelianViolation
MultiThreadedErrorTracker
NGSPlatform
NamedTable
PathUtils
PathUtils.DirectoryFilter
PathUtils.ExtensionFilter
PathUtils.OrFilenameFilter
PathUtils.isGATKVersion
Line 101, Col 13: statement executed 284 times.


Chrome File Edit View History Bookmarks Window Help
Mon 9:41 AM Mark Depristo
Edit Discussion - GATK-For
GATK clover report
file:///Users/depristo/Desktop/broadLocal/GATK/unstable/clover_html/index.html
Google News Computers Broad General ecommerce XM Calendar Mail Other Bookmarks
GATK clover report
Clover Coverage Report
Dashboard
Coverage Reports
Coverage (Aggregate)
Test Code (Aggregate)
Test Results
Application Packages
net.sf.picard.reference (0%)
net.sf.samtools (0%)
org.broadinstitute.sting (5.1%)
org.broadinstitute.sting.commandline (1.1%)
org.broadinstitute.sting.gatk (0%)
org.broadinstitute.sting.gatk.arguments (0%)
org.broadinstitute.sting.gatk.contexts (0%)
org.broadinstitute.sting.gatk.datasources.providers (0%)
org.broadinstitute.sting.utils
Classes Tests Results
Class
AutoFormattingTime
BitSetUtils
ContigComparator
GenomeLoc
GenomeLocComparator
GenomeLocParser
GenomeLocParser.CachingSequenceDictionary
GenomeLocSortedSet
Haplotype
Haplotype.HaplotypeBaseComparator
HeapSizeMonitor
HeapSizeMonitor.MonitorRunnable
IndelUtils
LRUCache
LegacyReservoirDownsampler
MannWhitneyU
MannWhitneyU.DitheringComparator
MannWhitneyU.NumberedPairComparator
MathUtils
MathUtils.RunningAverage
MathUtils.comparator
Median
MendelianViolation
MultiThreadedErrorTracker
NGSPlatform
NamedTable
PathUtils
PathUtils.DirectoryFilter
PathUtils.ExtensionFilter
PathUtils.OrFilenameFilter
PathUtils.isGATKVersion

- Some of the methods are greyed out. This is because they are considered by our clover report as trivial getter/setter methods, and shouldn't be counted.

- Some methods have reasonably good test coverage, such as disjointP with thousands of tests.

- Some methods have some tests, but a very limited number, such as contiguousP which only has 2 tests. Now maybe that's enough, but it's worth thinking about whether 2 tests would really cover all of the test cases for this method.

- Some methods (such as intersect) have good coverage on some branches but no coverage on what looks like an important branch (the unmapped handling code).

- Some methods just don't have any tests at all (subtract), which is very dangerous if this method is an important one used throughout the GATK.

For methods with poor test coverage (branches or overall) I'd look into their uses, and try to answer a few questions:

- How widely used this is function? Is this method used at all? Perhaps it's just unused code that can be deleted. Perhaps its only used in one specific class, and it's not worth my time testing it (a dangerous statement, as basically any untested code can assumed to be broken now, or some point in the future). If it's widely used, I should design some unit tests for it.

- Are the uses simpler than the full code itself? Perhaps a simpler function can be extracted, and it tested.

If the code needs tests, I would design specific unit tests (or data providers that cover all possible cases) for these function. Once that newly-written code is in place, I would rerun the ant tasks above to get updated coverage information, and continue until I'm satisfied.

Collecting output #1341

Last updated on 2012-10-18 15:27:03

1. Analysis output overview

In theory, any class implementing the `OutputStream` interface. In practice, three types of classes are commonly used: `PrintStreams` for plain text files, `SAMFileWriters` for BAM files, and `VCFWriters` for VCF files.

2. PrintStream

To declare a basic `PrintStream` for output, use the following declaration syntax:

```
@Output
public PrintStream out;
```

And use it just as you would any other PrintStream:

```
out.println("Hello, world!");
```

By default, `@Output` streams prepopulate `fullName`, `shortName`, `required`, and `doc`. `required` in this context means that the GATK will always fill in the contents of the `out` field for you. If the user specifies no `--out` command-line argument, the 'out' field will be prepopulated with a stream pointing to `System.out`.

If your walker outputs a custom format that requires more than simple concatenation by Queue you should also implement a custom `Gatherer`.

3. SAMFileWriter

For some applications, you might need to manage their own SAM readers and writers directly from inside your walker. Current best practice for creating these Readers / Writers is to declare arguments of type `SAMFileReader` or `SAMFileWriter` as in the following example:

```
@Output
SAMFileWriter outputBamFile = null;
```

If you do not specify the full name and short name, the writer will provide system default names for these arguments. Creating a `SAMFileWriter` in this way will create the type of writer most commonly used by members of the GSA group at the Broad Institute -- it will use the same header as the input BAM and require presorted data. To change either of these attributes, use the `StingSAMIterator` interface instead:

```
@Output
StingSAMFileWriter outputBamFile = null;
```

and later, in `initialize()`, run one or both of the following methods:

outputBAMFile.writeHeader(customHeader); outputBAMFile.setPresorted(false);

You can change the header or presorted state until the first alignment is written to the file.

4. VCFWriter

`VCFWriter` outputs behave similarly to `PrintStreams` and `SAMFileWriters`. Declare a `VCFWriter` as follows:

@Output(doc="File to which variants should be written",required=true) protected VCFWriter writer = null;

5. Debugging Output

The walkers provide a protected logger instance. Users can adjust the debug level of the walkers using the `-l` command line option.

Turning on verbose logging can produce more output than is really necessary. To selectively turn on logging for a class or package, specify a `log4j.properties` property file from the command line as follows:

```
-Dlog4j.configuration=file:///<your development root>/Sting/java/config/log4j.properties
```

An example `log4j.properties` file is available in the `java/config` directory of the Git repository.

Documenting walkers #1346

Last updated on 2013-03-25 18:25:50

The GATK discovers walker documentation by reading it out of the Javadoc, Sun's design pattern for providing documentation for packages and classes. This page will provide an extremely brief explanation of how to write Javadoc; more information on writing javadoc comments can be found in Sun's documentation.

1. Adding walker and package descriptions to the help text

The GATK's build system uses the javadoc parser to extract the javadoc for classes and packages and embed the contents of that javadoc in the help system. If you add Javadoc to your package or walker, it will automatically appear in the help. The javadoc parser will pick up on 'standard' javadoc comments, such as the following, taken from PrintReadsWalker:

```
/**
 * This walker prints out the input reads in SAM format.  Alternatively, the walker can
write reads into a specified BAM file.
 */
```

You can add javadoc to your package by creating a special file, `package-info.java`, in the package directory. This file should consist of the javadoc for your package plus a package descriptor line. One such example follows:

```
/**
 * @help.display.name Miscellaneous walkers (experimental)
 */
package org.broadinstitute.sting.playground.gatk.walkers;
```

Additionally, the GATK provides two extra custom tags for overriding the information that ultimately makes it into the help.

- `@help.display.name` Changes the name of the package as it appears in help. Note that the name of the walker cannot be changed as it is required to be passed verbatim to the `-T` argument.
- `@help.summary` Changes the description which appears on the right-hand column of the help text. This is useful if you'd like to provide a more concise description of the walker that should appear in the help.
- `@help.description` Changes the description which appears at the bottom of the help text with `-T <your walker> --help` is specified. This is useful if you'd like to present a more complete description of your walker.

2. Hiding experimental walkers (use sparingly, please!)

Walkers can be hidden from the documentation system by adding the `@Hidden` annotation to the top of each walker. `@Hidden` walkers can still be run from the command-line, but their documentation will not be visible to end users. Please use this functionality sparingly to avoid walkers with hidden command-line options that are required for production use.

3. Disabling building of help

Because the building of our help text is actually heavyweight and can dramatically increase compile time on some systems, we have a mechanism to disable help generation.

Compile with the following command:

```
ant -Ddisable.help=true
```

to disable generation of help.

Frequently asked questions about QScripts #1314

Last updated on 2012-10-18 15:38:17

1. Many of my GATK functions are setup with the same Reference, Intervals, etc. Is there a quick way to reuse these values for the different analyses in my pipeline?

Yes.

- Create a trait that extends from CommandLineGATK.
- In the trait, copy common values from your qscript.
- Mix the trait into instances of your classes.

For more information, see the `ExampleUnifiedGenotyper.scala` or examples of using Scala's traits/mixins illustrated in the QScripts documentation.

2. How do I accept a list of arguments to my QScript?

In your QScript, define a `var` list and annotate it with `@Argument`. Initialize the value to `Nil`.

```
@Argument(doc="filter names", shortName="filter")
var filterNames: List[String] = Nil
```

On the command line specify the arguments by repeating the argument name.

```
-filter filter1 -filter filter2 -filter filter3
```

Then once your QScript is run, the command line arguments will be available for use in the QScript's `script`

method.

```
  def script {
     var myCommand = new MyFunction
     myCommand.filters = this.filterNames
  }
```

For a full example of command line arguments see the QScripts documentation.

3. What is the best way to run a utility method at the right time?

Wrap the utility with an `InProcessFunction`. If your functionality is reusable code you should add it to `Sting Utils` with `Unit Tests` and then invoke your new function from your `InProcessFunction`. Computationally or memory intensive functions should NOT be implemented as `InProcessFunctions`, and should be wrapped in Queue CommandLineFunctions instead.

```
    class MySplitter extends InProcessFunction {
      @Input(doc="inputs")
      var in: File = _

      @Output(doc="outputs")
      var out: List[File] = Nil

      def run {
         StingUtilityMethod.quickSplitFile(in, out)
      }
    }

    var splitter = new MySplitter
    splitter.in = new File("input.txt")
    splitter.out = List(new File("out1.txt"), new File("out2.txt"))
    add(splitter)
```

See Queue CommandLineFunctions for more information on how `@Input` and `@Output` are used.

4. What is the best way to write a list of files?

Create an instance of a `ListWriterFunction` and add it in your script method.

```
import org.broadinstitute.sting.queue.function.ListWriterFunction
</pre>

<pre>
    val writeBamList = new ListWriterFunction
    writeBamList.inputFiles = bamFiles
    writeBamList.listFile = new File("myBams.list")
    add(writeBamList)
```

5. How do I add optional debug output to my QScript?

Queue contains a trait mixin you can use to add Log4J support to your classes.

Add the import for the trait `Logging` to your QScript.

```
import org.broadinstitute.sting.queue.util.Logging
```

Mixin the trait to your class.

```
class MyScript extends Logging {
...
```

Then use the mixed in `logger` to write debug output when the user specifies `-l DEBUG`.

```
logger.debug("This will only be displayed when debugging is enabled.")
```

6. I updated Queue and now I'm getting java.lang.NoClassDefFoundError / java.lang.AbstractMethodError

Try `ant clean`.

Queue relies on a lot of Scala traits / mixins. These dependencies are not always picked up by the scala/java compilers leading to partially implemented classes. If that doesn't work please let us know in the forum.

7. Do I need to create directories in my QScript?

No. QScript will create all parent directories for outputs.

8. How do I specify the -W 240 for the LSF hour queue at the Broad?

Queue's LSF dispatcher automatically looks up and sets the maximum runtime for whichever LSF queue is specified. If you set your `-jobQueue/.jobQueue` to `hour` then you should see something like this under `bjobs -l`:

```
RUNLIMIT
240.0 min of gsa3
```

9. Can I run Queue with GridEngine?

Queue GridEngine functionality is community supported. See here for full details: Queue with Grid Engine.

10. How do I pass advanced java arguments to my GATK commands, such as remote debugging?

The easiest way to do this at the moment is to mixin a trait.

First define a trait which adds your java options:

```
  trait RemoteDebugging extends JavaCommandLineFunction {
    override def javaOpts = super.javaOpts + " -Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005"
  }
```

Then mix in the trait to your walker and otherwise run it as normal:

```
  val printReadsDebug = new PrintReads with RemoteDebugging
  printReadsDebug.reference_sequence = "my.fasta"
  // continue setting up your walker...
  add(printReadsDebug)
```

11. Why does Queue log "Running jobs. ... Done." but doesn't actually run anything?

If you see something like the following, it means that Queue believes that it previously successfully generated all of the outputs.

```
INFO 16:25:55,049 QCommandLine - Scripting ExampleUnifiedGenotyper
INFO 16:25:55,140 QCommandLine - Added 4 functions
INFO 16:25:55,140 QGraph - Generating graph.
INFO 16:25:55,164 QGraph - Generating scatter gather jobs.
INFO 16:25:55,714 QGraph - Removing original jobs.
INFO 16:25:55,716 QGraph - Adding scatter gather jobs.
INFO 16:25:55,779 QGraph - Regenerating graph.
INFO 16:25:55,790 QGraph - Running jobs.
INFO 16:25:55,853 QGraph - 0 Pend, 0 Run, 0 Fail, 10 Done
INFO 16:25:55,902 QCommandLine - Done
```

Queue will not re-run the job if a `.done` file is found for the all the outputs, e.g.: `/path/to/.output.file.done`. You can either remove the specific `.done` files yourself, or use the `-startFromScratch` command line option.

Frequently asked questions about Scala #1315

Last updated on 2013-03-25 18:27:55

1. What is Scala?

Scala is a combination of an object oriented framework and a functional programming language. For a good introduction see the free online book Programming Scala.

The following are extremely brief answers to frequently asked questions about Scala which often pop up when first viewing or editing QScripts. For more information on Scala there a multitude of resources available around the web including the Scala home page and the online Scala Doc.

2. Where do I learn more about Scala?

- http://www.scala-lang.org
- http://programming-scala.labs.oreilly.com
- http://www.scala-lang.org/docu/files/ScalaByExample.pdf
- http://devcheatsheet.com/tag/scala/
- http://davetron5000.github.com/scala-style/index.html

3. What is the difference between `var` and `val`?

`var` is a value you can later modify, while `val` is similar to `final` in Java.

4. What is the difference between Scala collections and Java collections? / Why do I get the error: type mismatch?

Because the GATK and Queue are a mix of Scala and Java sometimes you'll run into problems when you need a Scala collection and instead a Java collection is returned.

```
MyQScript.scala:39: error: type mismatch;
  found   : java.util.List[java.lang.String]
  required: scala.List[String]
     val wrapped: List[String] = TextFormattingUtils.wordWrap(text, width)
```

Use the implicit definitions in `JavaConversions` to automatically convert the basic Java collections to and from Scala collections.

```
import collection.JavaConversions._
```

Scala has a very rich collections framework which you should take the time to enjoy. One of the first things you'll notice is that the default Scala collections are immutable, which means you should treat them as you would a String. When you want to 'modify' an immutable collection you need to capture the result of the operation, often assigning the result back to the original variable.

```
var str = "A"
str + "B"
println(str) // prints: A
str += "C"
println(str) // prints: AC

var set = Set("A")
set + "B"
println(set) // prints: Set(A)
set += "C"
println(set) // prints: Set(A, C)
```

5. How do I append to a list?

Use the `:+` operator for a single value.

```
var myList = List.empty[String]
myList :+= "a"
myList :+= "b"
myList :+= "c"
```

Use ++ for appending a list.

```
var myList = List.empty[String]
myList ++= List("a", "b", "c")
```

6. How do I add to a set?

Use the + operator.

```
var mySet = Set.empty[String]
mySet += "a"
mySet += "b"
mySet += "c"
```

7. How do I add to a map?

Use the + and `->` operators.

```
var myMap = Map.empty[String,Int]
myMap += "a" -> 1
myMap += "b" -> 2
myMap += "c" -> 3
```

8. What are Option, Some, and None?

Option is a Scala generic type that can either be some generic value or `None`. Queue often uses it to represent primitives that may be null.

```
var myNullableInt1: Option[Int] = Some(1)
var myNullableInt2: Option[Int] = None
```

9. What is _ / What is the underscore?

FranÃ§ois Armand's slide deck is a good introduction: http://www.slideshare.net/normation/scala-dreaded

To quote from his slides:

```
Give me a variable name but
- I don't care of what it is
- and/or
```

```
- don't want to pollute my namespace with it
```

10. How do I format a String?

Use the `.format()` method.

This Java snippet:

```
String formatted = String.format("%s %i", myString, myInt);
```

In Scala would be:

```
val formatted = "%s %i".format(myString, myInt)
```

11. Can I use Scala Enumerations as QScript @Arguments?

No. Currently Scala's `Enumeration` class does not interact with the Java reflection API in a way that could be used for Queue command line arguments. You can use Java `enums` if for example you are importing a Java based walker's `enum` type.

If/when we find a workaround for Queue we'll update this entry. In the meantime try using a String.

Frequently asked questions about using IntelliJ IDEA #1316

Last updated on 2012-10-18 15:37:02

1. Can I use the free IntelliJ IDEA Community Edition to work with Scala and Queue?

Yes. Be sure to install the scala plugin and setup your IDE as listed in [Queue with IntelliJ IDEA(http://gatkforums.broadinstitute.org/discussion/1309/queue-with-intellij-idea).

2. I updated IntelliJ IDEA and lost the ability to use command completion

Check if there is an update to your Scala plugin as well.

3. I can't compile Queue in IntelliJ IDEA / My Scala files are not highlighted correctly

Check your IntelliJ IDEA settings to for the following:

- The Scala plugin is installed
- Under `File Types` have `*.scala` as a registered pattern for `Scala files`.

GATK development process and coding standards #2129

Last updated on 2013-03-25 18:28:58

Introduction

This document describes the current GATK coding standards for documentation and unit testing. The overall goal is that all functions be well documented, have unit tests, and conform to the coding conventions described in this guideline. It is primarily meant as an internal reference for team members, but we are making it public to provide an example of how we work. There are a few mentions of specific team member responsibilities and who to contact with questions; please just disregard those as they will not be applicable to you.

Coding conventions

General conventions

The Genome Analysis Toolkit generally follows Java coding standards and good practices, which can be viewed at Sun's site.

The original coding standard document for the GATK was developed in early 2009. It remains a reasonable starting point but may be superseded by statements on this page (available as a PDF).

Size of functions and functional programming style

Code in the GATK should be structured into clear, simple, and testable functions. Clear means that the function takes a limited number of arguments, most of which are values not modified, and in general should return newly allocated results, as opposed to directly modifying the input arguments (functional style). The max. size of functions should be approximately one screen's worth of real estate (no more than 80 lines), including inline comments. If you are writing functions that are much larger than this, you must refactor your code into modular components.

Code duplication

Do not duplicate code. If you are finding yourself wanting to make a copy of functionality, refactor the code you want to duplicate and enhance it. Duplicating code introduces bugs, makes the system harder to maintain, and will require more work since you will have a new function that must be tested, as opposed to expanding the tests on the existing functionality.

Documentation

Functions must be documented following the javadoc conventions. That means that the first line of the comment should be a simple statement of the purpose of the function. Following that is an expanded description of the function, such as edge case conditions, requirements on the argument, state changes, etc. Finally comes the @param and @return fields, that should describe the meaning of each function argument, restrictions on the values allowed or returned. In general, the return field should be about types and ranges of those values, not the meaning of the result, as this should be in the body of the documentation.

Testing for valid inputs and contracts

The GATK uses Contracts for Java to help us enforce code quality during testing. See CoFoJa for more information. If you've never programmed with contracts, read their excellent description Adding contracts to a stack. Contracts are only enabled when we are testing the code (unittests and integration tests) and not during normal execution, so contracts can be reasonably expensive to compute. They are best used to enforce assumptions about the status of class variables and return results.

Contracts are tricky when it comes to input arguments. The best practice is simple:

- Public functions with arguments should explicitly test those input arguments for good values with live java code (such as in the example below). Because the function is public, you don't know what the caller will be passing in, so you have to check and ensure quality.
- Private functions with arguments should use contracts instead. Because the function is private, the author of the code controls use of the function, and the contracts enforce good use. But in principal the quality of the inputs should be assumed at runtime since only the author controlled calls to the function and input QC should have happened elsewhere

Below is an example private function that makes good use of input argument contracts:

```
/**
 * Helper function to write out a IGV formatted line to out, at loc, with values
 *
 * http://www.broadinstitute.org/software/igv/IGV
 *
 * @param out a non-null PrintStream where we'll write our line
 * @param loc the location of values
 * @param featureName string name of this feature (see IGV format)
 * @param values the floating point values to associate with loc and feature name in out
 */
@Requires({
        "out != null",
        "loc != null",
        "values.length > 0"
})
private void printIGVFormatRow(final PrintStream out, final GenomeLoc loc, final String
featureName, final double ... values) {
    // note that start and stop are 0 based, but the stop is exclusive so we don't subtract
1
    out.printf("%s\t%d\t%d\t%s", loc.getContig(), loc.getStart() - 1, loc.getStop(),
featureName);
    for ( final double value : values )
        out.print(String.format("\t%.3f", value));
    out.println();
}
```

Final variables

Final java fields cannot be reassigned once set. Nearly all variables you write should be final, unless they are obviously accumulator results or other things you actually want to modify. Nearly all of your function arguments should be final. Being final stops incorrect reassigns (a major bug source) as well as more clearly captures the flow of information through the code.

An example high-quality GATK function

```
/**
 * Get the reference bases from referenceReader spanned by the extended location of this
active region,
 * including additional padding bp on either side.  If this expanded region would exceed the
boundaries
 * of the active region's contig, the returned result will be truncated to only include
on-genome reference
 * bases
 * @param referenceReader the source of the reference genome bases
 * @param padding the padding, in BP, we want to add to either side of this active region
extended region
 * @param genomeLoc a non-null genome loc indicating the base span of the bp we'd like to
get the reference for
 * @return a non-null array of bytes holding the reference bases in referenceReader
 */
@Ensures("result != null")
public byte[] getReference( final IndexedFastaSequenceFile referenceReader, final int
padding, final GenomeLoc genomeLoc ) {
    if ( referenceReader == null ) throw new IllegalArgumentException("referenceReader
cannot be null");
    if ( padding < 0 ) throw new IllegalArgumentException("padding must be a positive
integer but got " + padding);
    if ( genomeLoc == null ) throw new IllegalArgumentException("genomeLoc cannot be null");
    if ( genomeLoc.size() == 0 ) throw new IllegalArgumentException("GenomeLoc must have
size > 0 but got " + genomeLoc);

    final byte[] reference =  referenceReader.getSubsequenceAt( genomeLoc.getContig(),
            Math.max(1, genomeLoc.getStart() - padding),

Math.min(referenceReader.getSequenceDictionary().getSequence(genomeLoc.getContig()).getSeque
nceLength(), genomeLoc.getStop() + padding) ).getBases();

    return reference;
}
```

Unit testing

All classes and methods in the GATK should have unit tests to ensure that they work properly, and to protect

yourself and others who may want to extend, modify, enhance, or optimize you code. That GATK development team assumes that anything that isn't unit tested is broken. Perhaps right now they aren't broken, but with a team of 10 people they will become broken soon if you don't ensure they are correct going forward with unit tests.

Walkers are a particularly complex issue. UnitTesting the map and reduce results is very hard, and in my view largely unnecessary. That said, you should write your walkers and supporting classes in such a way that all of the complex data processing functions are separated from the map and reduce functions, and those should be unit tested properly.

Code coverage tells you how much of your class, at the statement or function level, has unit testing coverage. The GATK development standard is to reach something >80% method coverage (and ideally >80% statement coverage). The target is flexible as some methods are trivial (they just call into another method) so perhaps don't need coverage. At the statement level, you get deducted from 100% for branches that check for things that perhaps you don't care about, such as illegal arguments, so reaching 100% statement level coverage is unrealistic for most clases.

You can find out more information about generating code coverage results at Analyzing coverage with clover

We've created a unit testing example template in the GATK codebase that provides examples of creating core GATK data structures from scratch for unit testing. The code is in class ExampleToCopyUnitTest and can be viewed here in github directly ExampleToCopyUnitTest.

The GSA-Workflow

As of GATK 2.5, we are moving to a full code review process, which has the following benefits:

- Reducing obvious coding bugs seen by other eyes
- Reducing code duplication, as reviewers will be able to see duplicated code within the commit and potentially across the codebase
- Ensure that coding quality standards are met (style and unit testing)
- Setting a higher code quality standard for the master GATK unstable branch
- Providing detailed coding feedback to newer developers, so they can improve their skills over time

The GSA workflow in words :

- Create a new branch to start any work. Never work on master.
- branch names have to follow the convention of [author prefix][feature name][JIRA ticket] (e.g. rp_pairhmm_GSA-232)

- Make frequent commits.
- Push frequently your branch to origin (branch -> branch)
- When you're done -- rewrite your commit history to tell a compelling story Git Tools Rewriting History
- Push your rewritten history, and request a code review.

- The entire GSA team will review your code

- Mark DePristo assigns the reviewer responsible for making the judgment based on all reviews and merge your code into master.

- If your pull-request gets rejected, follow the comments from the team to fix it and repeat the workflow until you're ready to submit a new pull request.

- If your pull-request is accepted, the reviewer will merge and remove your remote branch.

Example GSA workflow in the command line:

```
# starting a new feature
git checkout -b rp_pairhmm_GSA-332
git commit -av
git push -u origin rp_pairhmm_GSA-332

# doing work on existing feature
git commit -av
git push

# ready to submit pull-request
git fetch origin
git rebase -i origin/master
git push -f

# after being accepted, delete your branch
git checkout master
git pull
git branch -d rp_pairhmm_GSA-332
(the reviewer will remove your github branch)
```

Commit histories and rebasing

You must commit your code in small commit blocks with commit messages that follow the git best practices, which require the first line of the commit to summarize the purpose of the commit, followed by -- lines that describe the changes in more detail. For example, here's a recent commit that meets this criteria that added unit tests to the GenomeLocParser:

```
Refactoring and unit testing GenomeLocParser

-- Moved previously inner class to MRUCachingSAMSequenceDictionary, and unit test to 100%
coverage
-- Fully document all functions in GenomeLocParser
-- Unit tests for things like parsePosition (shocking it wasn't tested!)
-- Removed function to specifically create GenomeLocs for VariantContexts.  The fact that
you must incorporate END attributes in the context means that createGenomeLoc(Feature) works
correctly
-- Depreciated (and moved functionality) of setStart, setStop, and incPos to GenomeLoc
```

```
-- Unit test coverage at like 80%, moving to 100% with next commit
```

Now, git encourages you to commit code often, and develop your code in whatever order or what is best for you. So it's common to end up with 20 commits, all with strange, brief commit messages, that you want to push into the master branch. It is not acceptable to push such changes. You need to use the git command rebase to reorganize your commit history so satisfy the small number of clear commits with clear messages.

Here is a recommended git workflow using rebase:

- Start every project by creating a new branch for it. From your master branch, type the following command (replacing "myBranch" with an appropriate name for the new branch):

```
git checkout -b myBranch
```

Note that you only include the -b when you're first creating the branch. After a branch is already created, you can switch to it by typing the checkout command without the -b: "git checkout myBranch"

Also note that since you're always starting a new branch from master, you should keep your master branch up-to-date by occasionally doing a "git pull" while your master branch is checked out. You shouldn't do any actual work on your master branch, however.

- When you want to update your branch with the latest commits from the central repo, type this while your branch is checked out:

```
git fetch && git rebase origin/master
```

If there are conflicts while updating your branch, git will tell you what additional commands to use.

If you need to combine or reorder your commits, add "-i" to the above command, like so:

```
git fetch && git rebase -i origin/master
```

If you want to edit your commits without also retrieving any new commits, omit the "git fetch" from the above command.

If you find the above commands cumbersome or hard to remember, create aliases for them using the following commands:

```
    git config --global alias.up '!git fetch && git rebase origin/master'
    git config --global alias.edit '!git fetch && git rebase -i origin/master'
    git config --global alias.done '!git push origin HEAD:master'
```

Then you can type "git up" to update your branch, "git edit" to combine/reorder commits, and "git done" to push your branch.

Here are more useful tutorials on how to use rebase:

- Git Tools Rewriting History
- Keeping commit histories clean
- The case for git rebase
- Squashing commits with rebase

If you need help with rebasing, talk to Mauricio or David and they will help you out.

How to access the picard-public repository (containing sam-jdk, picard, tribble, and variant) #2194

The picard-public repository on sourceforge, in addition to housing the sam-jdk and picard-public, is now home to tribble and the org.broadinstitute.variant package (which includes VariantContext and associated classes as well as the VCF/BCF codecs).

If you just need to check out the sources and don't need to make any commits into the picard repository, the command is:

```
svn checkout svn://svn.code.sf.net/p/picard/code/trunk picard-public
```

Then you can attach the `picard-public/src/java` directory in IntelliJ as a source directory (File -> Project Structure -> Libraries -> Click the plus sign -> "Attach Files or Directories" in the latest IntelliJ).

To build picard, sam-jdk, tribble, and org.broadinstitute.variant all at once, type `ant` from within the picard-public directory. To run tests, type `ant test`

If you do need to make commits into the picard repository, first you'll need to create a sourceforge account, then contact the Picard team and request access for that account. Once you've been given access, you'll need to check out their repository using a command of this form:

```
svn checkout --username=YOUR_USERNAME
svn+ssh://YOUR_USERNAME@svn.code.sf.net/p/picard/code/trunk picard-public
```

How to make a walker compatible with multi-threading #2867

Last updated on 2013-06-26 19:05:10

This document provides an overview of what are the steps required to make a walker multi-threadable using the `-nct` and the `-nt` arguments, which make use of the `NanoSchedulable` and `TreeReducible` interfaces, respectively.

NanoSchedulable / `-nct`

Providing `-nct` support requires that you certify that your walker's `map()` method is thread-safe -- eg., if any data structures are shared across `map()` calls, access to these must be properly synchronized. Once your `map()` method is thread-safe, you can implement the `NanoSchedulable` interface, an empty interface with no methods that just marks your walker as having a `map()` method that's safe to parallelize:

```
/**
 * Root parallelism interface.  Walkers that implement this
 * declare that their map function is thread-safe and so multiple
 * map calls can be run in parallel in the same JVM instance.
 */
public interface NanoSchedulable {
}
```

TreeReducible / `-nt`

Providing `-nt` support requires that both `map()` and `reduce()` be thread-safe, and you also need to implement the `TreeReducible` interface. Implementing `TreeReducible` requires you to write a `treeReduce()` method that tells the engine how to combine the results of multiple `reduce()` calls:

```
public interface TreeReducible<ReduceType> {
    /**
     * A composite, 'reduce of reduces' function.
     * @param lhs 'left-most' portion of data in the composite reduce.
     * @param rhs 'right-most' portion of data in the composite reduce.
     * @return The composite reduce type.
     */
    ReduceType treeReduce(ReduceType lhs, ReduceType rhs);
}
```

This method differs from `reduce()` in that while `reduce()` adds the result of a single `map()` call onto a running total, `treeReduce()` takes the aggregated results from multiple map/reduce tasks that have been run in parallel and combines them. So, `lhs` and `rhs` might each represent the final result from several hundred map/reduce calls.

Example `treeReduce()` implementation from the UnifiedGenotyper:

```
public UGStatistics treeReduce(UGStatistics lhs, UGStatistics rhs) {
    lhs.nBasesCallable += rhs.nBasesCallable;
    lhs.nBasesCalledConfidently += rhs.nBasesCalledConfidently;
    lhs.nBasesVisited += rhs.nBasesVisited;
    lhs.nCallsMade += rhs.nCallsMade;
    return lhs;
}
```

Managing user inputs #1325

Last updated on 2012-10-18 15:34:05

1. Naming walkers

Users identify which GATK walker to run by specifying a walker name via the `--analysis_type` command-line argument. By default, the GATK will derive the walker name from a walker by taking the name of the walker class and removing packaging information from the start of the name, and removing the trailing text `Walker` from the end of the name, if it exists. For example, the GATK would, by default, assign the name `PrintReads` to the walker class `org.broadinstitute.sting.gatk.walkers.PrintReadsWalker`. To override the default walker name, annotate your walker class with `@WalkerName("<my name>")`.

2. Requiring / allowing primary inputs

Walkers can flag exactly which primary data sources are allowed and required for a given walker. Reads, the reference, and reference-ordered data are currently considered primary data sources. Different traversal types have different default requirements for reads and reference, but currently no traversal types require reference-ordered data by default. You can add requirements to your walker with the `@Requires` / `@Allows` annotations as follows:

```
@Requires(DataSource.READS)
@Requires({DataSource.READS,DataSource.REFERENCE})
@Requires(value={DataSource.READS,DataSource.REFERENCE})
@Requires(value=DataSource.REFERENCE})
```

By default, all parameters are allowed unless you lock them down with the `@Allows` attribute. The command:

```
@Allows(value={DataSource.READS,DataSource.REFERENCE})
```

will only allow the reads and the reference. Any other primary data sources will cause the system to exit with an error.

Note that as of August 2011, the GATK no longer supports RMD the `@Requires` and `@Allows` syntax, as these have moved to the standard `@Argument` system.

3. Command-line argument tagging

Any command-line argument can be tagged with a comma-separated list of freeform tags.

The syntax for tags is as follows:

```
-<argument>:<tag1>,<tag2>,<tag3> <argument value>
```

for example:

```
-I:tumor <my tumor data>.bam
```

```
-eval,VCF yri.trio.chr1.vcf
```

There is currently no mechanism in the GATK to validate either the number of tags supplied or the content of those tags.

Tags can be accessed from within a walker by calling `getToolkit().getTags(argumentValue)`, where `argumentValue` is the parsed contents of the command-line argument to inspect.

Applications

The GATK currently has comprehensive support for tags on two built-in argument types:

`--I,--input_file <input_file>`

Input BAM files and BAM file lists can be tagged with any type. When a BAM file list is tagged, the tag is applied to each listed BAM file.

From within a walker, use the following code to access the supplied tag or tags:

```
getToolkit().getReaderIDForRead(read).getTags();
```

- Input RODs, e.g. `-V ' or '-eval '

Tags are used to specify ROD name and ROD type. There is currently no support for adding additional tags. See the ROD system documentation for more details.

4. Adding additional command-line arguments

Users can create command-line arguments for walkers by creating public member variables annotated with `@Argument` in the walker. The `@Argument` annotation takes a number of differentparameters:

- `fullName`

The full name of this argument. Defaults to the `toLowerCase()`â€™d member name. When specifying `fullName` on the command line, prefix with a double dash (`--`).

- `shortName`

The alternate, short name for this argument. Defaults to the first letter of the member name. When specifying shortName on the command line, prefix with a single dash (`-`).

- `doc`

Documentation for this argument. Will appear in help output when a user either requests help with the â€“-help (-h) argument or when a user specifies an invalid set of arguments. Documentation is the only argument that is always required.

- `required`

Whether the argument is required when used with this walker. Default is `required = true`.

- `exclusiveOf`

Specifies that this argument is mutually exclusive of another argument in the same walker. Defaults to not mutually exclusive of any other arguments.

- `validation`

Specifies a regular expression used to validate the contents of the command-line argument. If the text provided by the user does not match this regex, the GATK will abort with an error.

By default, all command-line arguments will appear in the help system. To prevent new and debugging arguments from appearing in the help system, you can add the `@Hidden` tag below the `@Argument` annotation, hiding it from the help system but allowing users to supply it on the command-line. Please use this functionality sparingly to avoid walkers with hidden command-line options that are required for production use.

Passing Command-Line Arguments

Arguments can be passed to the walker using either the full name or the short name. If passing arguments using the full name, the syntax is `â^'â^'<arg full name> <value>`.

```
--myint 6
```

If passing arguments using the short name, the syntax is `-<arg short name> <value>`. Note that there is a space between the short name and the value:

```
-m 6
```

Boolean (class) and boolean (primitive) arguments are a special in that they require no argument. The presence of a boolean indicates true, and its absence indicates false. The following example sets a flag to true.

```
-B
```

Supplemental command-line argument annotations

Two additional annotations can influence the behavior of command-line arguments.

- `@Hidden`

Adding this annotation to an @Argument tells the help system to avoid displaying any evidence that this argument exists. This can be used to add additional debugging arguments that aren't suitable for mass consumption.

- `@Deprecated`

Forces the GATK to throw an exception if this argument is supplied on the command-line. This can be used to supply extra documentation to the user as command-line parameters change for walkers that are in flux.

Examples

Create an required int parameter with full name â€“myint, short name -m. Pass this argument by adding â€“myint 6 or -m 6 to the command line.

```
import org.broadinstitute.sting.utils.cmdLine.Argument;
public class HelloWalker extends ReadWalker<Integer,Long> {
    @Argument(doc="my integer")
    public int myInt;
```

Create an optional float parameter with full name â€“myFloatingPointArgument, short name -m. Pass this argument by adding â€“myFloatingPointArgument 2.71 or -m 2.71.

```
import org.broadinstitute.sting.utils.cmdLine.Argument;
public class HelloWalker extends ReadWalker<Integer,Long> {
    @Argument(fullName="myFloatingPointArgument",doc="a floating point
argument",required=false)
    public float myFloat;
```

The GATK will parse the argument differently depending on the type of the public member variableâ€™s type. Many different argument types are supported, including primitives and their wrappers, arrays, typed and untyped collections, and any type with a String constructor. When the GATK cannot completely infer the type (such as in the case of untyped collections), it will assume that the argument is a String. GATK is aware of concrete implementations of some interfaces and abstract classes. If the argumentâ€™s member variable is of type List or Set, the GATK will fill the member variable with a concrete ArrayList or TreeSet, respectively. Maps are not currently supported.

5. Additional argument types: @Input, @Output

Besides @Argument, the GATK provides two additional types for command-line arguments: @Input and @Output. These two inputs are very similar to @Argument but act as flags to indicate dataflow to Queue, our pipeline management software.

- The @Input tag indicates that the contents of the tagged field represents a file that will be read by the walker.
- The @Output tag indicates that the contents of the tagged field represents a file that will be written by the walker, for consumption by downstream walkers.

We're still determining the best way to model walker dependencies in our pipeline. As we determine best practices, we'll post them here.

6. Getting access to Reference Ordered Data (RMD) with @Input and RodBinding

As of August 2011, the GATK now provides a clean mechanism for creating walker @Input arguments and using these arguments to access Reference Meta Data provided by the RefMetaDataTracker in the map() call. This mechanism is preferred to the old implicit string-based mechanism, which has been retired.

At a very high level, the new RodBindings provide a handle for a walker to obtain the Feature records from

Tribble from a map() call, specific to a command line binding provided by the user. This can be as simple as a single ROD file argument|one-to-one binding between a command line argument and a track, or as complex as an argument argument accepting multiple command line arguments, each with a specific name. The RodBindings are generic and type specific, so you can require users to provide files that emit VariantContexts, BedTables, etc, or simply the root type Feature from Tribble. Critically, the RodBindings interact nicely with the GATKDocs system, so you can provide summary and detailed documentation for each RodBinding accepted by your walker.

A single ROD file argument

Suppose you have a walker that uses a single track of VariantContexts, such as SelectVariants, in its calculation. You declare a standard GATK-style @Input argument in the walker, of type RodBinding< VariantContext>:

```
@Input(fullName="variant", shortName = "V", doc="Select variants from this VCF file",
required=true)
public RodBinding<VariantContext> variants;
```

This will require the user to provide a command line option --variant:vcf my.vcf to your walker. To get access to your variants, in the map() function you provide the variants variable to the tracker, as in:

```
Collection<VariantContext> vcs = tracker.getValues(variants, context.getLocation());
```

which returns all of the VariantContexts in variants that start at context.getLocation(). See RefMetaDataTracker in the javadocs to see the full range of getter routines.

Note that, as with regular tribble tracks, you have to provide the Tribble type of the file as a tag to the argument (:vcf). The system now checks up front that the corresponding Tribble codec produces Features that are type-compatible with the type of the RodBinding<T>.

RodBindings are generic

The RodBinding class is generic, parameterized as RodBinding<T extends Feature>. This T class describes the type of the Feature required by the walker. The best practice for declaring a RodBinding is to choose the most general Feature type that will allow your walker to work. For example, if all you really care about is whether a Feature overlaps the site in map, you can use Feature itself, which supports this, and will allow any Tribble type to be provided, using a RodBinding<Feature>. If you are manipulating VariantContexts, you should declare a RodBinding<VariantContext>, which will restrict automatically the user to providing Tribble types that can create a object consistent with the VariantContext class (a VariantContext itself or subclass).

Note that in multi-argument RodBindings, as List<RodBinding<T>> arg, the system will require all files provided here to provide an object of type T. So List<RodBinding<VariantContext>> arg requires all -arg command line arguments to bind to files that produce VariantContexts.

An argument that can be provided any number of times

The RodBinding system supports the standard @Argument style of allowing a vararg argument by wrapping it in a Java collection. For example, if you want to allow users to provide any number of comp tracks to your walker, simply declare a List<RodBinding<VariantContext>> field:

```
@Input(fullName="comp", shortName = "comp", doc="Comparison variants from this VCF file",
required=true)
public List<RodBinding<VariantContext>> comps;
```

With this declaration, your walker will accept any number of -comp arguments, as in:

```
-comp:vcf 1.vcf -comp:vcf 2.vcf -comp:vcf 3.vcf
```

For such a command line, the comps field would be initialized to the List with three RodBindings, the first binding to 1.vcf, the second to 2.vcf and finally the third to 3.vcf.

Because this is a required argument, at least one -comp must be provided. Vararg @Input RodBindings can be optional, but you should follow proper varargs style to get the best results.

Proper handling of optional arguments

If you want to make a RodBinding optional, you first need to tell the @Input argument that its options (required=false):

```
@Input(fullName="discordance", required=false)
private RodBinding<VariantContext> discordanceTrack;
```

The GATK automagically sets this field to the value of the special static constructor method makeUnbound(Class c) to create a special "unbound" RodBinding here. This unbound object is type safe, can be safely passed to the RefMetaDataTracker get methods, and is guaranteed to never return any values. It also returns false when the isBound() method is called.

An example usage of isBound is to conditionally add header lines, as in:

```
if ( mask.isBound() ) {
    hInfo.add(new VCFFilterHeaderLine(MASK_NAME, "Overlaps a user-input mask"));
}
```

The case for vararg style RodBindings is slightly different. If you want, as above, users to be able to omit the -comp track entirely, you should initialize the value of the collection to the appropriate emptyList/emptySet in Collections:

```
@Input(fullName="comp", shortName = "comp", doc="Comparison variants from this VCF file",
required=false)
public List<RodBinding<VariantContext>> comps = Collections.emptyList();
```

which will ensure that `comps.isEmpty()` is true when no `-comp` is provided.

Implicit and explicit names for RodBindings

```
@Input(fullName="variant", shortName = "V", doc="Select variants from this VCF file",
required=true)
public RodBinding<VariantContext> variants;
```

By default, the `getName()` method in `RodBinding` returns the `fullName` of the `@Input`. This can be overloaded on the command-line by providing not one but two tags. The first tag is interpreted as the name for the binding, and the second as the type. As in:

```
-variant:vcf foo.vcf     => getName() == "variant"
     -variant:foo,vcf foo.vcf => getName() == "foo"
```

This capability is useful when users need to provide more meaningful names for arguments, especially with variable arguments. For example, in `VariantEval`, there's a `List<RodBinding<VariantContext>>` comps, which may be `dbsnp`, `hapmap`, etc. This would be declared as:

```
@Input(fullName="comp", shortName = "comp", doc="Comparison variants from this VCF file",
required=true)
public List<RodBinding<VariantContext>> comps;
```

where a normal command line usage would look like:

```
-comp:hapmap,vcf hapmap.vcf -comp:omni,vcf omni.vcf -comp:1000g,vcf 1000g.vcf
```

In the code, you might have a loop that looks like:

```
for ( final RodBinding comp : comps )
    for ( final VariantContext vc : tracker.getValues(comp, context.getLocation())
        out.printf("%s has a binding at %s%n", comp.getName(),
getToolkit().getGenomeLocParser.createGenomeLoc(vc));
```

which would print out lines that included things like:

```
hapmap has a binding at 1:10
omni has a binding at 1:20
hapmap has a binding at 1:30
1000g has a binding at 1:30
```

This last example begs the question -- what happens with `getName()` when explicit names are not provided? The system goes out of its way to provide reasonable names for the variables:

- The first occurrence is named for the `fullName`, where `comp`
- Subsequent occurrences are postfixed with an integer count, starting at 2, so `comp2`, `comp3`, etc.

In the above example, the command line

```
-comp:vcf hapmap.vcf -comp:vcf omni.vcf -comp:vcf 1000g.vcf
```

would emit

```
comp has a binding at 1:10
comp2 has a binding at 1:20
comp has a binding at 1:30
comp3 has a binding at 1:30
```

Dynamic type resolution

The new `RodBinding` system supports a simple form of dynamic type resolution. If the input filetype can be specially associated with a single `Tribble` type (as VCF can), then you can omit the type entirely from the the command-line binding of a `RodBinding`!

So whereas a full command line would look like:

```
-comp:hapmap,vcf hapmap.vcf -comp:omni,vcf omni.vcf -comp:1000g,vcf 1000g.vcf
```

because these are VCF files they could technically be provided as:

```
-comp:hapmap hapmap.vcf -comp:omni omni.vcf -comp:1000g 1000g.vcf
```

If you don't care about naming, you can now say:

```
-comp hapmap.vcf -comp omni.vcf -comp 1000g.vcf
```

Best practice for documenting a RodBinding

The best practice is simple: use a javadoc style comment above the `@Input` annotation, with the standard first line summary and subsequent detailed discussion of the meaning of the argument. These are then picked up by the GATKdocs system and added to the standard walker docs, following the standard structure of GATKDocs `@Argument` docs. Below is a best practice documentation example from `SelectVariants`, which accepts a required variant track and two optional discordance and concordance tracks.

```
public class SelectVariants extends RodWalker<Integer, Integer> {
   /**
     * Variants from this file are sent through the filtering and modifying routines as
directed
     * by the arguments to SelectVariants, and finally are emitted.
     */
    @Input(fullName="variant", shortName = "V", doc="Select variants from this VCF file",
required=true)
    public RodBinding<VariantContext> variants;

    /**
```

```
     * A site is considered discordant if there exists some sample in eval that has a
non-reference genotype
     * and either the site isn't present in this track, the sample isn't present in this
track,
     * or the sample is called reference in this track.
     */
    @Input(fullName="discordance", shortName = "disc", doc="Output variants that were not
called in this Feature comparison track", required=false)
    private RodBinding<VariantContext> discordanceTrack;

    /**
     * A site is considered concordant if (1) we are not looking for specific samples and
there is a variant called
     * in both variants and concordance tracks or (2) every sample present in eval is
present in the concordance
     * track and they have the sample genotype call.
     */
    @Input(fullName="concordance", shortName = "conc", doc="Output variants that were also
called in this Feature comparison track", required=false)
    private RodBinding<VariantContext> concordanceTrack;
}
```

Note how much better the above version is compared to the old pre-`Rodbinding` syntax (code below). Below you have a required argument variant that doesn't show up as a formal argument in the GATK, different from the conceptually similar `@Arguments` for `discordanceRodName` and `concordanceRodName`, which have no type restrictions. There's no place to document the variant argument as well, so the system is effectively blind to this essential argument.

```
@Requires(value={},referenceMetaData=@RMD(name="variant", type=VariantContext.class))
public class SelectVariants extends RodWalker<Integer, Integer> {
    @Argument(fullName="discordance", shortName =  "disc", doc="Output variants that were
not called on a ROD comparison track. Use -disc ROD_NAME", required=false)
    private String discordanceRodName = "";

    @Argument(fullName="concordance", shortName =  "conc", doc="Output variants that were
also called on a ROD comparison track. Use -conc ROD_NAME", required=false)
    private String concordanceRodName = "";
}
```

RodBinding examples

In these examples, we have declared two `RodBindings` in the Walker

```
@Input(fullName="mask", doc="Input ROD mask", required=false)
public RodBinding<Feature> mask = RodBinding.makeUnbound(Feature.class);

@Input(fullName="comp", doc="Comparison track", required=false)
```

```
public List<RodBinding<VariantContext>> comps = new ArrayList<VariantContext>();
```

- Get the first value

```
Feature f = tracker.getFirstValue(mask)
```

- Get all of the values at a location

```
Collection<Feature> fs = tracker.getValues(mask, thisGenomeLoc)
```

- Get all of the features here, regardless of track

```
Collection<Feature> fs = tracker.getValues(Feature.class)
```

- Determining if an optional RodBinding was provided . if (mask.isBound()) // writes out the mask header line, if one was provided hInfo.add(new VCFFilterHeaderLine(MASK_NAME, "Overlaps a user-input mask"));

if (! comps.isEmpty()) logger.info("At least one comp was provided")

Example usage in Queue scripts

In QScripts when you need to tag a file use the class `TaggedFile` which extends from `java.io.File`.

Example	in the QScript	on the Command Line
Untagged VCF	myWalker.variant = new File("my.vcf")	-V my.vcf
Tagged VCF	myWalker.variant = new TaggedFile("my.vcf", "VCF")	-V:VCF my.vcf
Tagged VCF	myWalker.variant = new TaggedFile("my.vcf", "VCF,custom=value")	-V:VCF,custom=value my.vcf
Labeling a tumor	myWalker.input_file :+= new TaggedFile("mytumor.bam", "tumor")	-I:tumor mytumor.bam

Notes

No longer need to (or can) use `@Requires` and `@Allows` for ROD data. This system is now retired.

Managing walker data presentation and flow control #1351

Last updated on 2012-10-18 15:20:32

The primary goal of the GATK is to provide a suite of small data access patterns that can easily be parallelized and otherwise externally managed. As such, rather than asking walker authors how to iterate over a data stream, the GATK asks the user how data should be presented.

Locus walkers

Walk over the data set one location (single-base locus) at a time, presenting all overlapping reads, reference bases, and reference-ordered data.

1. Switching between covered and uncovered loci

The @By attribute can be used to control whether locus walkers see all loci or just covered loci. To switch between viewing all loci and covered loci, apply one of the following attributes:

```
@By(DataSource.REFERENCE)
@By(DataSource.READS)
```

2. Filtering defaults

By default, the following filters are automatically added to every locus walker.

- Reads with nonsensical alignments
- Unmapped reads
- Non-primary alignments.
- Duplicate reads.
- Reads failing vendor quality checks.

ROD walkers

These walkers walk over the data set one location at a time, but only those locations covered by reference-ordered data. They are essentially a special case of locus walkers. ROD walkers are read-free traversals that include operate over Reference Ordered Data and the reference genome **at sites where there is ROD information**. They are geared for high-performance traversal of many RODs and the reference such as VariantEval and CallSetConcordance. Programmatically they are nearly identical to `RefWalkers<M,T>` traversals with the following few quirks.

1. Differences from a RefWalker

- RODWalkers are only called at sites where there is at least one non-interval ROD bound. For example, if you are exploring dbSNP and some GELI call set, the map function of a RODWalker will be invoked at all sites where there is a dbSNP record or a GELI record.
- Because of this skipping RODWalkers receive a context object where the number of reference skipped bases between map calls is provided:

nSites += context.getSkippedBases() + 1; // the skipped bases plus the current location

In order to get the final count of skipped bases at the end of an interval (or chromosome) the map function is called one last time with null `ReferenceContext` and `RefMetaDataTracker` objects. The alignment context can be accessed to get the bases skipped between the last (and final) ROD and the end of the current interval.

2. Filtering defaults

ROD walkers inherit the same filters as locus walkers:

- Reads with nonsensical alignments

- Unmapped reads
- Non-primary alignments.
- Duplicate reads.
- Reads failing vendor quality checks.

3. Example change over of VariantEval

Changing to a RODWalker is very easy -- here's the new top of VariantEval, changing the system to a `RodWalker` from its old `RefWalker` state:

```
//public class VariantEvalWalker extends RefWalker<Integer, Integer> {
public class VariantEvalWalker extends RodWalker<Integer, Integer> {
```

The map function must now capture the number of skipped bases and protect itself from the final interval map calls:

```
public Integer map(RefMetaDataTracker tracker, ReferenceContext ref, AlignmentContext
context) {
    nMappedSites += context.getSkippedBases();

    if ( ref == null ) { // we are seeing the last site
        return 0;
    }

    nMappedSites++;
```

That's all there is to it!

4. Performance improvements

A ROD walker can be very efficient compared to a RefWalker in the situation where you have sparse RODs. Here is a comparison of ROD vs. Ref walker implementation of VariantEval:

	RODWalker	RefWalker
dbSNP and 1KG Pilot 2 SNP calls on chr1	164u (s)	768u (s)
Just 1KG Pilot 2 SNP calls on chr1	54u (s)	666u (s)

Read walkers

Read walkers walk over the data set one read at a time, presenting all overlapping reference bases and reference-ordered data.

Filtering defaults

By default, the following filters are automatically added to every read walker.

- Reads with nonsensical alignments

Read pair walkers

Read pair walkers walk over a queryname-sorted BAM, presenting each mate and its pair. No reference bases or reference-ordered data are presented.

Filtering defaults

By default, the following filters are automatically added to every read pair walker.

- Reads with nonsensical alignments

Duplicate walkers

Duplicate walkers walk over a read and all its marked duplicates. No reference bases or reference-ordered data are presented.

Filtering defaults

By default, the following filters are automatically added to every duplicate walker.

- Reads with nonsensical alignments
- Unmapped reads
- Non-primary alignments

Output management #1327

Last updated on 2012-10-18 15:32:05

1. Introduction

When running either single-threaded or in shared-memory parallelism mode, the GATK guarantees that output written to an output stream created via the `@Argument` mechanism will ultimately be assembled in genomic order. In order to assemble the final output file, the GATK will write the output generated from each thread into a temporary output file, ultimately assembling the data via a central coordinating thread. There are three major elements in the GATK that facilitate this functionality:

- Stub

  The front-end interface to the output management system. Stubs will be injected into the walker by the command-line argument system and relay information from the walker to the output management system. There will be one stub per invocation of the GATK.

- Storage

The back end interface, responsible for creating, writing and deleting temporary output files as well as merging their contents back into the primary output file. One Storage object will exist per shard processed in the GATK.

- OutputTracker

The dispatcher; ultimately connects the stub object's output creation request back to the most appropriate storage object to satisfy that request. One OutputTracker will exist per GATK invocation.

2. Basic Mechanism

Stubs are directly injected into the walker through the GATK's command-line argument parser as a go-between from walker to output management system. When a walker calls into the stub it's first responsibility is to call into the output tracker to retrieve an appropriate storage object. The behavior of the OutputTracker from this point forward depends mainly on the parallelization mode of this traversal of the GATK.

If the traversal is single-threaded:

- the OutputTracker (implemented as DirectOutputTracker) will create the storage object if necessary and return it to the stub.
- The stub will forward the request to the provided storage object.
- At the end of the traversal, the microscheduler will request that the OutputTracker finalize and close the file.

If the traversal is multi-threaded using shared-memory parallelism:

- The OutputTracker (implemented as ThreadLocalOutputTracker) will look for a storage object associated with this thread via a ThreadLocal.
- If no such storage object exists, it will be created pointing to a temporary file.
- At the end of **each shard processed**, that file will be closed and an OutputMergeTask will be created so that the shared-memory parallelism code can merge the output at its leisure.
- The shared-memory parallelism code will merge when a fixed number of temporary files appear in the input queue. The constant used to determine this frequency is fixed at compile time (see `HierarchicalMicroScheduler.MAX_OUTSTANDING_OUTPUT_MERGES`).

3. Using output management

To use the output management system, declare a field in your walker of one of the existing core output types, coupled with either an `@Argument` or `@Output` annotation.

```
@Output(doc="Write output to this BAM filename instead of STDOUT")
SAMFileWriter out;
```

Currently supported output types are SAM/BAM (declare SAMFileWriter), VCF (declare VCFWriter), and any non-buffering stream extending from OutputStream.

4. Implementing a new output type

To create a new output type, three types must be implemented: Stub, Storage, and ArgumentTypeDescriptor.

To implement Stub

Create a new Stub class, extending/inheriting the core output type's interface and implementing the Stub interface.

```
OutputStreamStub extends OutputStream implements Stub<OutputStream> {
```

Implement a register function so that the engine can provide the stub with the session's OutputTracker.

```
public void register( OutputTracker outputTracker ) {
    this.outputTracker = outputTracker;
}
```

Add as fields any parameters necessary for the storage object to create temporary storage.

```
private final File targetFile;
public File getOutputFile() { return targetFile; }
```

Implement/override every method in the core output type's interface to pass along calls to the appropriate storage object via the OutputTracker.

```
public void write( byte[] b, int off, int len ) throws IOException {
    outputTracker.getStorage(this).write(b, off, len);
}
```

To implement Storage

Create a Storage class, again extending inheriting the core output type's interface and implementing the Storage interface.

```
public class OutputStreamStorage extends OutputStream implements Storage<OutputStream> {
```

Implement constructors that will accept just the Stub or Stub + alternate file path and create a repository for data, and a close function that will close that repository.

```
public OutputStreamStorage( OutputStreamStub stub ) { ... }
public OutputStreamStorage( OutputStreamStub stub, File file ) { ... }
public void close() { ... }
```

Implement a `mergeInto` function capable of reconstituting the file created by the constructor, dumping it back into the core output type's interface, and removing the source file.

```
public void mergeInto( OutputStream targetStream ) { ... }
```

Add a block to `StorageFactory.createStorage()` capable of creating the new storage object. **TODO: use reflection to generate the storage classes.**

```
    if(stub instanceof OutputStreamStub) {
        if( file != null )
            storage = new OutputStreamStorage((OutputStreamStub)stub,file);
        else
            storage = new OutputStreamStorage((OutputStreamStub)stub);
    }
```

To implement ArgumentTypeDescriptor

Create a new object inheriting from type `ArgumentTypeDescriptor`. Note that the `ArgumentTypeDescriptor` does NOT need to support the core output type's interface.

```
public class OutputStreamArgumentTypeDescriptor extends ArgumentTypeDescriptor {
```

Implement a truth function indicating which types this `ArgumentTypeDescriptor` can service.

```
 @Override
 public boolean supports( Class type ) {
     return SAMFileWriter.class.equals(type) || StingSAMFileWriter.class.equals(type);
 }
```

Implement a parse function that constructs the new Stub object. The function should register this type as an output by caling `engine.addOutput(stub)`.

```
 public Object parse( ParsingEngine parsingEngine, ArgumentSource source, Type type,
ArgumentMatches matches )  {
     ...
     OutputStreamStub stub = new OutputStreamStub(new File(fileName));
     ...
     engine.addOutput(stub);
     ....
     return stub;
}
```

Add a creator for this new ArgumentTypeDescriptor in `CommandLineExecutable.getArgumentTypeDescriptors()`.

```
 protected Collection<ArgumentTypeDescriptor> getArgumentTypeDescriptors() {
     return Arrays.asList( new
VCFWriterArgumentTypeDescriptor(engine,System.out,argumentSources),
                           new SAMFileWriterArgumentTypeDescriptor(engine,System.out),
                           new OutputStreamArgumentTypeDescriptor(engine,System.out) );
 }
```

After creating these three objects, the new output type should be ready for usage as described above.

5. Outstanding issues

- Only non-buffering iterators are currently supported by the GATK. Of particular note, `PrintWriter` will appear to drop records if created by the command-line argument system; use `PrintStream` instead.

- For efficiency, the GATK does not reduce output files together following the tree pattern used by shared-memory parallelism; output merges happen via an independent queue. Because of this, output merges happening during a `treeReduce` may not behave correctly.

Overview of Queue #1306

Last updated on 2013-09-04 20:55:00

1. Introduction

GATK-Queue is command-line scripting framework for defining multi-stage genomic analysis pipelines combined with an execution manager that runs those pipelines from end-to-end. Often processing genome data includes several steps to produces outputs, for example our BAM to VCF calling pipeline include among other things:

- Local realignment around indels
- Emitting raw SNP calls
- Emitting indels
- Masking the SNPs at indels
- Annotating SNPs using chip data
- Labeling suspicious calls based on filters
- Creating a summary report with statistics

Running these tools one by one in series may often take weeks for processing, or would require custom scripting to try and optimize using parallel resources.

With a Queue script users can semantically define the multiple steps of the pipeline and then hand off the logistics of running the pipeline to completion. Queue runs independent jobs in parallel, handles transient errors, and uses various techniques such as running multiple copies of the same program on different portions of the genome to produce outputs faster.

2. Obtaining Queue

You have two options: donwload the binary distribution (prepackaged, ready to run program) or build it from source.

- Download the binary

This is obviously the easiest way to go. Links are on the Downloads page.

- Building Queue from source

Briefly, here's what you need to know/do:

Queue is part of the Sting repository. Download the source from the public repository on Github. Run the following command:

```
git clone git://github.com/broadgsa/gatk.git Sting
```

IMPORTANT NOTE: These instructions refer to the MIT-licensed version of the GATK+Queue source code. With that version, you will be able to build Queue itself, as well as the public portion of the GATK (the core framework), but that will not include the GATK analysis tools. If you want to use Queue to pipeline the GATK analysis tools, you need to clone the 'protected' repository. Please note however that part of the source code in that repository (the 'protected' module) is under a different license which excludes for-profit use, modification and redistribution.

Use ant to build the source.

```
cd Sting
ant queue
```

Queue uses the Ivy dependency manager to fetch all other dependencies. Just make sure you have suitable versions of the JDK and Ant!

See this article on how to test your installation of Queue.

3. Running Queue

See this article on running Queue for the first time for full details.

Queue arguments can be listed by running with `--help`

java -jar dist/Queue.jar --help

To list the arguments required by a QScript, add the script with `-S` and run with `--help`.

java -jar dist/Queue.jar -S script.scala --help

Note that by default queue runs in a "dry" mode, as explained in the link above. After verifying the generated commands execute the pipeline by adding `-run`.

See QFunction and Command Line Options for more info on adjusting Queue options.

4. QScripts

General Information

Queue pipelines are written as Scala 2.8 files with a bit of syntactic sugar, called QScripts.

Every QScript includes the following steps:

- New instances of CommandLineFunctions are created
- Input and output arguments are specified on each function
- The function is added with `add()` to Queue for dispatch and monitoring

The basic command-line to run the Queue pipelines on the command line is

```
java -jar Queue.jar -S <script>.scala
```

See the main article Queue QScripts for more info on QScripts.

Supported QScripts

Most QScripts are analysis pipelines that are custom-built for specific projects, and we currently do not offer any QScripts as supported analysis tools. However, we do provide some example scripts that you can use as basis to write your own QScripts (see below).

Example QScripts

The latest version of the example files are available in the Sting github repository under public/scala/qscript/examples

5. Visualization and Queue

QJobReport

Queue automatically generates GATKReport-formatted runtime information about executed jobs. See this presentation for a general introduction to QJobReport.

Note that Queue attempts to generate a standard visualization using an R script in the GATK `public/R` repository. You must provide a path to this location if you want the script to run automatically. Additionally the script requires the `gsalib` to be installed on the machine, which is typically done by providing its path in your `.Rprofile` file:

bm8da-dbe ~/Desktop/broadLocal/GATK/unstable % cat ~/.Rprofile
.libPaths("/Users/depristo/Desktop/broadLocal/GATK/unstable/public/R/")

Caveats

- The system only provides information about commands that have just run. Resuming from a partially completed job will only show the information for the jobs that just ran, and not for any of the completed commands. This is due to a structural limitation in Queue, and will be fixed when the Queue infrastructure improves
- This feature only works for command line and LSF execution models. SGE should be easy to add for a motivated individual but we cannot test this capabilities here at the Broad. Please send us a patch if you do extend Queue to support SGE.

DOT visualization of Pipelines

Queue emits a `queue.dot` file to help visualize your commands. You can open this file in programs like DOT, OmniGraffle, etc to view your pipelines. By default the system will print out your LSF command lines, but this can be too much in a complex pipeline.

To clarify your pipeline, override the `dotString()` function:

```
class CountCovariates(bamIn: File, recalDataIn: File, args: String = "") extends
GatkFunction {
    @Input(doc="foo") var bam = bamIn
    @Input(doc="foo") var bamIndex = bai(bamIn)
    @Output(doc="foo") var recalData = recalDataIn
    memoryLimit = Some(4)
    override def dotString = "CountCovariates: %s [args %s]".format(bamIn.getName, args)
    def commandLine = gatkCommandLine("CountCovariates") + args + " -l INFO -D
/humgen/gsa-hpprojects/GATK/data/dbsnp_129_hg18.rod -I %s --max_reads_at_locus 20000 -cov
ReadGroupCovariate -cov QualityScoreCovariate -cov CycleCovariate -cov DinucCovariate
-recalFile %s".format(bam, recalData)
}
```

Here we only see `CountCovariates my.bam [-OQ]`, for example, in the `dot` file. The base quality score recalibration pipeline, as visualized by DOT, can be viewed here:

6. Further reading

- How to run Queue for the first time
- Queue with IntelliJ IDEA
- Queue QScripts
- QFunction and Command Line Options
- Queue CommandLineFunctions
- Pipelining the GATK using Queue
- Queue with Grid Engine
- Queue Frequently Asked Questions

Packaging and redistributing walkers #1301

Last updated on 2012-10-31 15:01:13

1. Redistributing the GATK-Lite or distributing walkers

The GATK team would love to hear about any applications within which of the GATK-Lite codebase is embedded or walkers which you have chosen to distribute. Please send an email to gsahelp to let us know!

When redistributing the GATK-Lite codebase, please abide by the terms of our copyright:

```
/*
 * Copyright (c) 2009 The Broad Institute
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use,
 * copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following
 * conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 */
```

2. Packaging walkers

The packaging tool in the Sting repository can layout packages for redistribution. Currently, only walkers checked into the GATK's git repository are well supported by the packaging system. Example packaging files can be found in `$STING_HOME/packages`.

3. Defining a package

Create a package xml for your project inside `$STING_HOME/packages`.

Key elements within the package xml include:

- `executable`

  Each occurrence of this tag will create an executable jar of the given name tag, using the main method from the given `main-class` tag.

- `main-class`

  This is the main class for the package. When running with `java -jar YOUR_JAR.jar`, `main-class` is the class that will be executed.

- `dependencies`

Other dependencies can be of type class or file. If of type `class`, a dependency analyzer will look for all dependencies of your classes and include those files as well. File dependencies will end up in the root of your package.

- `resources`

Supplemental files can be added to the `resources` section. Resource files will be copied to the `resources` directory within the package.

3. Creating a package

To create a package, execute the following command:

```
cd $STING_HOME
ant package -Dexecutable=<your executable name>
```

The packaging system will create a layout directory in `dist/packages/<your executable>`. Examine the contents of this directory. When you are happy with the results, finalize the package by running the following:

```
tar cvhjf <your executable>.tar.bz2 <your executable>
```

Pipelining the GATK with Queue #1310

Last updated on 2012-10-18 15:11:39

1. Introduction

As mentioned in the introductory materials, the core concept behind the GATK tools is the walker. The Queue scripting framework contains several mechanisms which make it easy to chain together GATK walkers.

2. Authoring walkers

As part of authoring your walker there are several Queue behaviors that you can specify for QScript authors using your particular walker.

Specifying how to partition

Queue can significantly speed up generating walker outputs by passing different instances of the GATK the same BAM or VCF data but specifying different regions of the data to analyze. After the different instances output their individual results Queue will gather the results back to the original output path requested by QScript.

Queue limits the level it will split genomic data by examining the `@PartitionBy()` annotation for your walker which specifies a `PartitionType`. This table lists the different partition types along with the default partition level for each of the different walker types.

PartitionType	Default for Walker Type	Description	Example Intervals	Example Splits
PartitionType.CONTIG	Read walkers	Data is grouped together so that all genomic data from the same contig is never presented to two different instances of the GATK.	original: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60, chr3:10-11	split 1: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60; split 2:chr3:10-11
PartitionType.INTERVAL	(none)	Data is split down to the interval level but never divides up an explicitly specified interval. If no explicit intervals are specified in the QScript for the GATK then this is effectively the same as splitting by contig.	original: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60, chr3:10-11	split 1: chr1:10-11, chr2:10-20, chr2:30-40; split 2: chr2:50-60, chr3:10-11
PartitionType.LOCUS	Locus walkers, ROD walkers	Data is split down to the locus level possibly dividing up intervals.	original: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60, chr3:10-11	split 1: chr1:10-11, chr2:10-20, chr2:30-35; split 2: chr2:36-40, chr2:50-60, chr3:10-11
PartitionType.NONE	Read pair walkers, Duplicate walkers	The data cannot be split and Queue must run the single instance of the GATK as specified in the QScript.	original: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60, chr3:10-11	no split: chr1:10-11, chr2:10-20, chr2:30-40, chr2:50-60, chr3:10-11

If you walker is implemented in a way that Queue should not divide up your data you should explicitly set the `@PartitionBy(PartitionType.NONE)`. If your walker can theoretically be run per genome location specify `@PartitionBy(PartitionType.LOCUS)`.

```
@PartitionBy(PartitionType.LOCUS)
public class ExampleWalker extends LocusWalker<Integer, Integer> {
...
```

Specifying how to join outputs

Queue will join the standard walker outputs.

Output type	Default gatherer implementation
SAMFileWriter	The BAM files are joined together using Picard's MergeSamFiles.
VCFWriter	The VCF files are joined together using the GATK CombineVariants.
PrintStream	The first two files are scanned for a common header. The header is written once into the output, and then each file is appended to the output, skipping past with the header lines.

If your PrintStream is not a simple text file that can be concatenated together, you must implement a `Gatherer`. Extend your custom Gatherer from the abstract base class and implement the `gather()` method.

```
package org.broadinstitute.sting.commandline;

import java.io.File;
import java.util.List;

/**
 * Combines a list of files into a single output.
 */
public abstract class Gatherer {
    /**
     * Gathers a list of files into a single output.
     * @param inputs Files to combine.
     * @param output Path to output file.
     */
    public abstract void gather(List<File> inputs, File output);

    /**
     * Returns true if the caller should wait for the input files to propagate over NFS
before running gather().
     */
    public boolean waitForInputs() { return true; }
}
```

Specify your gatherer using the `@Gather()` annotation by your `@Output`.

```
@Output
@Gather(MyGatherer.class)
public PrintStream out;
```

Queue will run your custom gatherer to join the intermediate outputs together.

3. Using GATK walkers in Queue

Queue GATK Extensions

Running 'ant queue' builds a set of Queue extensions for the GATK-Engine. Every GATK walker and command line program in the compiled `GenomeAnalysisTK.jar` a Queue compatible wrapper is generated.

The extensions can be imported via `import org.broadinstitute.sting.queue.extensions.gatk._`

```
import org.broadinstitute.sting.queue.QScript
import org.broadinstitute.sting.queue.extensions.gatk._

class MyQscript extends QScript {
...
```

Note that the generated GATK extensions will automatically handle shell-escaping of all values assigned to the various Walker parameters, so you can rest assured that all of your values will be taken literally by the shell. Do **not** attempt to escape values yourself -- ie.,

Do this:

```
filterSNPs.filterExpression = List("QD<2.0", "MQ<40.0", "HaplotypeScore>13.0")
```

NOT this:

```
filterSNPs.filterExpression = List("\"QD<2.0\"", "\"MQ<40.0\"", "\"HaplotypeScore>13.0\"")
```

Listing variables

In addition to the GATK documentation on thisÂ wiki you can also find the full list of arguments for each walker extension in a variety of ways.

The source code for the extensions is generated during `ant queue` and placed in this directory:

```
build/queue-extensions/src
```

When properly configured an IDE can provide command completion of the walker extensions. See Queue with IntelliJ IDEA for our recommended settings.

If you do not have access to an IDE you can still find the names of the generated variables using the command line. The generated variable names on each extension are based off of the `fullName` of the Walker argument. To see the built in documentation for each Walker, run the GATK with:

```
java -jar GenomeAnalysisTK.jar -T <walker name> -help
```

Once the import statement is specified you can add() instances of gatk extensions in your QScript's script() method.

Setting variables

If the GATK walker input allows more than one of a value you should specify the values as a `List()`.

```
  def script() {
    val snps = new UnifiedGenotyper
    snps.reference_file = new File("testdata/exampleFASTA.fasta")
    snps.input_file = List(new File("testdata/exampleBAM.bam"))
    snps.out = new File("snps.vcf")
    add(snps)
  }
```

Although it may be harder for others trying to read your QScript, for each of the long name arguments the

extensions contain aliases to their short names as well.

```
  def script() {
    val snps = new UnifiedGenotyper
    snps.R = new File("testdata/exampleFASTA.fasta")
    snps.I = List(new File("testdata/exampleBAM.bam"))
    snps.out = new File("snps.vcf")
    add(snps)
  }
```

Here are a few more examples using various list assignment operators.

```
  def script() {
    val countCovariates = new CountCovariates

    // Append to list using item appender :+
    countCovariates.rodBind :+= RodBind("dbsnp", "VCF", dbSNP)

    // Append to list using collection appender ++
    countCovariates.covariate ++= List("ReadGroupCovariate", "QualityScoreCovariate",
"CycleCovariate", "DinucCovariate")

    // Assign list using plain old object assignment
    countCovariates.input_file = List(inBam)

    // The following is not a list, so just assigning one file to another
    countCovariates.recal_file = outRecalFile

    add(countCovariates)
  }
```

Specifying an alternate GATK jar

By default Queue runs the GATK from the current classpath. This works best since the extensions are generated and compiled at time same time the GATK is compiled via `ant queue`.

If you need to swap in a different version of the GATK you may not be able to use the generated extensions. **The alternate GATK jar must have the same command line arguments as the GATK compiled with Queue.** Otherwise the arguments will not match and you will get an error when Queue attempts to run the alternate GATK jar. In this case you will have to create your own custom `CommandLineFunction` for your analysis.

```
  def script {
    val snps = new UnifiedGenotyper
    snps.jarFile = new File("myPatchedGATK.jar")
    snps.reference_file = new File("testdata/exampleFASTA.fasta")
    snps.input_file = List(new File("testdata/exampleBAM.bam"))
```

```
    snps.out = new File("snps.vcf")
    add(snps)
  }
```

GATK scatter/gather

Queue currently allows QScript authors to explicitly invoke scatter/gather on GATK walkers by setting the scatter count on a function.

```
  def script {
    val snps = new UnifiedGenotyper
    snps.reference_file = new File("testdata/exampleFASTA.fasta")
    snps.input_file = List(new File("testdata/exampleBAM.bam"))
    snps.out = new File("snps.vcf")
    snps.scatterCount = 20
    add(snps)
  }
```

This will run the UnifiedGenotyper up to 20 ways parallel and then will merge the partial VCFs back into the single `snps.vcf`.

Additional caveat

Some walkers are still being updated to support Queue fully. For example they may not have defined the `@Input` and `@Output` and thus Queue is unable to correctly track their dependencies, or a custom `Gatherer` may not be implemented yet.

QFunction and Command Line Options #1311

Last updated on 2012-10-18 15:13:31

These are the most popular Queue command line options. For a complete and up to date list run with `-help`. QScripts may also add additional command line options.

1. Queue Command Line Options

Command Line Argument	Description	Default
-run	If passed the scripts are run. If not passed a dry run is executed.	dry run
-jobRunner jobrunner	The job runner to dispatch jobs. Setting to Lsf706, GridEngine, or Drmaa will dispatch jobs to LSF or Grid Engine using the job settings (see below). Defaults to Shell which runs jobs on a local shell one at a time.	Shell
-bsub	Alias for -jobRunner Lsf706	not set
-qsub	Alias for -jobRunner GridEngine	not set

-status	Prints out a summary progress. If a QScript is currently running via -run, you can run the same command line with -status instead to print a summary of progress.	not set
-retry count	Retries a QFunction that returns a non-zero exit code up to count times. The QFunction must not have set jobRestartable to false.	0 = no retries
-startFromScratch	Restarts the graph from the beginning. If not specified for each output file specified on a QFunction, ex: pathtooutput.file, Queue will not re-run the job if a .done file is found for the all the outputs, ex: pathto.output.file.done.	use .done files to determine if jobs are complete
-keepIntermediates	By default Queue deletes the output files of QFunctions that set .isIntermediate to true.	delete intermediate files
-statusTo email	Email address to send status to whenever a) A job fails, or b) Queue has run all the functions it can run and is exiting.	not set
-statusFrom email	Email address to send status emails from.	user@local.domain
-dot file	If set renders the job graph to a dot file.	not rendered
-l logging_level	The minimum level of logging, DEBUG, INFO, WARN, or FATAL.	INFO
-log file	Sets the location to save log output in addition to standard out.	not set
-debug	Set the logging to include a lot of debugging information (SLOW!)	not set
-jobReport	Path to write the job report text file. If R is installed and available on the $PATH then a pdf will be generated visualizing the job report.	jobPrefix.jobreport.txt
-disableJobReport	Disables writing the job report.	not set
-help	Lists all of the command line arguments with their descriptions.	not set

2. QFunction Options

The following options can be specified on the command line over overridden per QFunction.

Command Line Argument	QFunction Property	Description	Default
-jobPrefix	.jobName	The unique name of the job. Used to prefix directories and log files. Use -jobNamePrefix on the Queue command line to replace the default prefix Q-processid@host.	jobNamePrefix-jobNumber
NA	.jobOutputFile	Captures stdout and if jobErrorFile is null it captures stderr as well.	jobName.out
NA	.jobErrorFile	If not null captures stderr.	null
NA	.commandDirectory	The directory to execute the command line from.	current directory
-jobProject	.jobProject	The project name for the job.	default job runner project
-jobQueue	.jobQueue	The queue to dispatch the job.	default job runner queue
-jobPriority	.jobPriority	The dispatch priority for the job. Lowest priority = 0. Highest priority = 100.	default job runner priority

-jobNative	.jobNativeArgs	Native args to pass to the job runner. Currently only supported in GridEngine and Drmaa. The string is concatenated to the native arguments passed over DRMAA. Example: -w n.	none
-jobResReq	.jobResourceRequests	Resource requests to pass to the job runner. On GridEngine this is multiple -l req. On LSF a single -R req is generated.	memory reservations and limits on LSF and GridEngine
-jobEnv	.jobEnvironmentNames	Predefined environment names to pass to the job runner. On GridEngine this is -pe env. On LSF this is -a env.	none
-memLimit	.memoryLimit	The memory limit for the job in gigabytes. Used to populate the variables residentLimit and residentRequest which can also be set separately.	default job runner memory limit
-resMemLimit	.residentLimit	Limit for the resident memory in gigabytes. On GridEngine this is -l mem_free=mem. On LSF this is -R rusage[mem=mem].	memoryLimit * 1.2
-resMemReq	.residentRequest	Requested amount of resident memory in gigabytes. On GridEngine this is -l h_rss=mem. On LSF this is -R rusage[select=mem].	memoryLimit

3. Email Status Options

Command Line Argument	Description	Default
-emailHost hostname	SMTP host name	localhost
-emailPort port	SMTP port	25
-emailTLS	If set uses TLS.	not set
-emailSSL	If set uses SSL.	not set
-emailUser username	If set along with emailPass or emailPassFile authenticates the email with this username.	not set
-emailPassFile file	If emailUser is also set authenticates the email with contents of the file.	not set
-emailPass password	If emailUser is also set authenticates the email with this password. NOT SECURE: Use emailPassFile instead!	not set

Queue CommandLineFunctions #1312

Last updated on 2012-10-18 15:40:00

1. Basic QScript run rules

- In the `script` method, a QScript will add one or more `CommandLineFunctions`.
- Queue tracks dependencies between functions via variables annotated with `@Input` and `@Output`.
- Queue will run functions based on the dependencies between them, so if the `@Input` of

`CommandLineFunction` A depends on the `@Output` of `ComandLineFunction` B, A will wait for B to finish before it starts running.

2. Command Line

Each CommandLineFunction must define the actual command line to run as follows.

```
class MyCommandLine extends CommandLineFunction {
  def commandLine = "myScript.sh hello world"
}
```

Constructing a Command Line Manually

If you're writing a one-off CommandLineFunction that is not destined for use by other QScripts, it's often easiest to construct the command line directly rather than through the API methods provided in the CommandLineFunction class.

For example:

```
def commandLine = "cat %s | grep -v \"#\" > %s".format(files, out)
```

Constructing a Command Line using API Methods

If you're writing a CommandLineFunction that will become part of Queue and/or will be used by other QScripts, however, our best practice recommendation is to construct your command line only using the methods provided in the CommandLineFunction class: `required()`, `optional()`, `conditional()`, and `repeat()`

The reason for this is that these methods automatically escape the values you give them so that they'll be interpreted literally within the shell scripts Queue generates to run your command, and they also manage whitespace separation of command-line tokens for you. This prevents (for example) a value like `MQ > 10` from being interpreted as an output redirection by the shell, and avoids issues with values containing embedded spaces. The methods also give you the ability to turn escaping and/or whitespace separation off as needed. An example:

```
override def commandLine = super.commandLine +
                           required("eff") +
                           conditional(verbose, "-v") +
                           optional("-c", config) +
                           required("-i", "vcf") +
                           required("-o", "vcf") +
                           required(genomeVersion) +
                           required(inVcf) +
                           required(">", escape=false) +  // This will be shell-interpreted
as an output redirection
                           required(outVcf)
```

The CommandLineFunctions built into Queue, including the CommandLineFunctions automatically generated for GATK Walkers, are all written using this pattern. This means that when you configure a GATK Walker or one of

the other built-in CommandLineFunctions in a QScript, you can rely on all of your values being safely escaped and taken literally when the commands are run, including values containing characters that would normally be interpreted by the shell such as `MQ > 10`.

Below is a brief overview of the API methods available to you in the `CommandLineFunction` class for safely constructing command lines:

- `required()`

Used for command-line arguments that are always present, e.g.:

```
required("-f", "filename")                              returns: " '-f' 'filename' "
required("-f", "filename", escape=false)                returns: " -f filename "
required("java")                                        returns: " 'java' "
required("INPUT=", "myBam.bam", spaceSeparated=false)   returns: " 'INPUT=myBam.bam' "
```

- `optional()`

Used for command-line arguments that may or may not be present, e.g.:

```
optional("-f", myVar) behaves like required() if myVar has a value, but returns ""
if myVar is null/Nil/None
```

- `conditional()`

Used for command-line arguments that should only be included if some condition is true, e.g.:

```
conditional(verbose, "-v") returns " '-v' " if verbose is true, otherwise returns ""
```

- `repeat()`

Used for command-line arguments that are repeated multiple times on the command line, e.g.:

```
repeat("-f", List("file1", "file2", "file3")) returns: " '-f' 'file1' '-f' 'file2' '-f'
'file3' "
```

3. Arguments

- `CommandLineFunction` arguments use a similar syntax to arguments.
- `CommandLineFunction` variables are annotated with `@Input`, `@Output`, or `@Argument` annotations.

Input and Output Files

So that Queue can track the input and output files of a command, `CommandLineFunction` `@Input` and `@Output` must be `java.io.File` objects.

```
class MyCommandLine extends CommandLineFunction {
  @Input(doc="input file")
```

```
  var inputFile: File = _
  def commandLine = "myScript.sh -fileParam " + inputFile
}
```

FileProvider

`CommandLineFunction` variables can also provide indirect access to `java.io.File` inputs and outputs via the `FileProvider` trait.

```
class MyCommandLine extends CommandLineFunction {
  @Input(doc="named input file")
  var inputFile: ExampleFileProvider = _
  def commandLine = "myScript.sh " + inputFile
}

// An example FileProvider that stores a 'name' with a 'file'.
class ExampleFileProvider(var name: String, var file: File) extends
org.broadinstitute.sting.queue.function.FileProvider {
  override def toString = " -fileName " + name + " -fileParam " + file
}
```

Optional Arguments

Optional files can be specified via `required=false`, and can use the `CommandLineFunction.optional()` utility method, as described above:

```
class MyCommandLine extends CommandLineFunction {
  @Input(doc="input file", required=false)
  var inputFile: File = _
  // -fileParam will only be added if the QScript sets inputFile on this instance of
MyCommandLine
  def commandLine = required("myScript.sh") + optional("-fileParam", inputFile)
}
```

Collections as Arguments

A `List` or `Set` of files can use the `CommandLineFunction.repeat()` utility method, as described above:

```
class MyCommandLine extends CommandLineFunction {
  @Input(doc="input file")
  var inputFile: List[File] = Nil // NOTE: Do not set List or Set variables to null!
  // -fileParam will added as many times as the QScript adds the inputFile on this instance
of MyCommandLine
  def commandLine = required("myScript.sh") + repeat("-fileParam", inputFile)
}
```

Non-File Arguments

A command line function can define other required arguments via @Argument.

```
class MyCommandLine extends CommandLineFunction {
  @Argument(doc="message to display")
  var veryImportantMessage: String = _
  // If the QScript does not specify the required veryImportantMessage, the pipeline will
not run.
  def commandLine = required("myScript.sh") + required(veryImportantMessage)
}
```

4. Example: "samtools index"

```
class SamToolsIndex extends CommandLineFunction {
  @Input(doc="bam to index") var bamFile: File = _
  @Output(doc="bam index") var baiFile: File = _
  def commandLine = "samtools index %s %s".format(bamFile, baiFile)
)
```

Or, using the CommandLineFunction API methods to construct the command line with automatic shell escaping:

```
class SamToolsIndex extends CommandLineFunction {
  @Input(doc="bam to index") var bamFile: File = _
  @Output(doc="bam index") var baiFile: File = _
  def commandLine = required("samtools") + required("index") + required(bamFile) +
required(baiFile)
)
```

Queue custom job schedulers #1347

Last updated on 2012-10-18 15:25:11

Implementing a Queue JobRunner

The following scala methods need to be implemented for a new JobRunner. See the implementations of GridEngine and LSF for concrete full examples.

1. class JobRunner.start()

Start should to copy the settings from the CommandLineFunction into your job scheduler and invoke the command via `sh <jobScript>`. As an example of what needs to be implemented, here is the current contents of the `start()` method in `MyCustomJobRunner` which contains the pseudo code.

```
  def start() {
    // TODO: Copy settings from function to your job scheduler syntax.

    val mySchedulerJob = new ...
```

```
// Set the display name to 4000 characters of the description (or whatever your max is)
mySchedulerJob.displayName = function.description.take(4000)

// Set the output file for stdout
mySchedulerJob.outputFile = function.jobOutputFile.getPath

// Set the current working directory
mySchedulerJob.workingDirectory = function.commandDirectory.getPath

// If the error file is set specify the separate output for stderr
if (function.jobErrorFile != null) {
  mySchedulerJob.errFile = function.jobErrorFile.getPath
}

// If a project name is set specify the project name
if (function.jobProject != null) {
  mySchedulerJob.projectName = function.jobProject
}

// If the job queue is set specify the job queue
if (function.jobQueue != null) {
  mySchedulerJob.queue = function.jobQueue
}

// If the resident set size is requested pass on the memory request
if (residentRequestMB.isDefined) {
  mySchedulerJob.jobMemoryRequest = "%dM".format(residentRequestMB.get.ceil.toInt)
}

// If the resident set size limit is defined specify the memory limit
if (residentLimitMB.isDefined) {
  mySchedulerJob.jobMemoryLimit = "%dM".format(residentLimitMB.get.ceil.toInt)
}

// If the priority is set (user specified Int) specify the priority
if (function.jobPriority.isDefined) {
  mySchedulerJob.jobPriority = function.jobPriority.get
}

// Instead of running the function.commandLine, run "sh <jobScript>"
mySchedulerJob.command = "sh " + jobScript

// Store the status so it can be returned in the status method.
myStatus = RunnerStatus.RUNNING

// Start the job and store the id so it can be killed in tryStop
```

```
    myJobId = mySchedulerJob.start()
  }
```

2. class JobRunner.status

The status method should return one of the enum values from `org.broadinstitute.sting.queue.engine.RunnerStatus`:

- `RunnerStatus.RUNNING`
- `RunnerStatus.DONE`
- `RunnerStatus.FAILED`

3. object JobRunner.init()

Add any initialization code to the companion object static initializer. See the LSF or GridEngine implementations for how this is done.

4. object JobRunner.tryStop()

The jobs that are still in `RunnerStatus.RUNNING` will be passed into this function. `tryStop()` should send these jobs the equivalent of a `Ctrl-C` or `SIGTERM(15)`, or worst case a `SIGKILL(9)` if `SIGTERM` is not available.

Running Queue with a new JobRunner

Once there is a basic implementation, you can try out the Hello World example with `-jobRunner MyJobRunner`.

```
java -Djava.io.tmpdir=tmp -jar dist/Queue.jar -S scala/qscript/examples/HelloWorld.scala
-jobRunner MyJobRunner -run
```

If all goes well Queue should dispatch the job to your job scheduler and wait until the status returns `RunningStatus.DONE` and `hello world` should be echo'ed into the output file, possibly with other log messages.

See QFunction and Command Line Options for more info on Queue options.

Queue pipeline scripts (QScripts) #1307

Last updated on 2012-10-18 15:15:47

1. Introduction

Queue pipelines are Scala 2.8 files with a bit of syntactic sugar, called QScripts. Check out the following as references.

- http://programming-scala.labs.oreilly.com

- http://www.scala-lang.org/docu/files/ScalaByExample.pdf
- http://davetron5000.github.com/scala-style/index.html

QScripts are easiest to develop using an Integrated Development Environment. See Queue with IntelliJ IDEA for our recommended settings.

The following is a basic outline of a QScript:

```
import org.broadinstitute.sting.queue.QScript
// List other imports here

// Define the overall QScript here.
class MyScript extends QScript {
  // List script arguments here.
  @Input(doc="My QScript inputs")
  var scriptInput: File = _

  // Create and add the functions in the script here.
  def script = {
     var myCL = new MyCommandLine
     myCL.myInput = scriptInput // Example variable input
     myCL.myOutput = new File("/path/to/output") // Example hardcoded output
     add(myCL)
  }

}
```

2. Imports

Imports can be any scala or java imports in scala syntax.

```
import java.io.File
import scala.util.Random
import org.favorite.my._
// etc.
```

3. Classes

- To add a `CommandLineFunction` to a pipeline, a class must be defined that extends `QScript`.
- The `QScript` must define a method `script`.
- The `QScript` can define helper methods or variables.

4. Script method

The body of `script` should create and add Queue CommandlineFunctions.

```
class MyScript extends org.broadinstitute.sting.queue.QScript {
```

```
  def script = add(new CommandLineFunction { def commandLine = "echo hello world" })
}
```

5. Command Line Arguments

- A `QScript` canbe set to read command line arguments by defining variables with `@Input`, `@Output`, or `@Argument` annotations.

- A command line argument can be a primitive scalar, enum, `File`, or scala immutable `Array`, `List`, `Set`, or `Option` of a primitive, enum, or `File`.

- `QScript` command line arguments can be marked as optional by setting `required=false`.

class MyScript extends org.broadinstitute.sting.queue.QScript { @Input(doc="example message to echo") var message: String = _ def script = add(new CommandLineFunction { def commandLine = "echo " + message }) }

6. Using and writing CommandLineFunctions

Adding existing GATK walkers

See Pipelining the GATK using Queue for more information on the automatically generated Queue wrappers for GATK walkers.

After functions are defined they should be added to the `QScript` pipeline using `add()`.

```
for (vcf <- vcfs) {
  val ve = new VariantEval
  ve.vcfFile = vcf
  ve.evalFile = swapExt(vcf, "vcf", "eval")
  add(ve)
}
```

Defining new CommandLineFunctions

- Queue tracks dependencies between functions via variables annotated with `@Input` and `@Output`.

- Queue will run functions based on the dependencies between them, not based on the order in which they are added in the script! So if the `@Input` of `CommandLineFunction` `A` depends on the `@Output` of `ComandLineFunction` `B`, `A` will wait for `B` to finish before it starts running.

- See the main article Queue CommandLineFunctions for more information.

7. Examples

- The latest version of the example files are available in the Sting git repository under public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/.

- To print the list of arguments required by an existing QScript run with `-help`.

- To check if your script has all of the `CommandLineFunction` variables set correctly, run without `-run`.

- When you are ready to execute the full pipeline, add `-run`.

Hello World QScript

The following is a "hello world" example that runs a single command line to `echo hello world`.

```
import org.broadinstitute.sting.queue.QScript

class HelloWorld extends QScript {
  def script = {
    add(new CommandLineFunction {
      def commandLine = "echo hello world"
    })
  }
}
```

The above file is checked into the Sting git repository under HelloWorld.scala. After building Queue from source, the QScript can be run with the following command:

```
java -Djava.io.tmpdir=tmp -jar dist/Queue.jar -S
public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/HelloWorld.scala -run
```

It should produce output similar to:

```
INFO  16:23:27,825 QScriptManager - Compiling 1 QScript
INFO  16:23:31,289 QScriptManager - Compilation complete
INFO  16:23:34,631 HelpFormatter - ---------------------------------------------------------

INFO  16:23:34,631 HelpFormatter - Program Name: org.broadinstitute.sting.queue.QCommandLine

INFO  16:23:34,632 HelpFormatter - Program Args: -S
public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/HelloWorld.scala -run

INFO  16:23:34,632 HelpFormatter - Date/Time: 2011/01/14 16:23:34
INFO  16:23:34,632 HelpFormatter - ---------------------------------------------------------

INFO  16:23:34,632 HelpFormatter - ---------------------------------------------------------

INFO  16:23:34,634 QCommandLine - Scripting HelloWorld
INFO  16:23:34,651 QCommandLine - Added 1 functions
INFO  16:23:34,651 QGraph - Generating graph.
INFO  16:23:34,660 QGraph - Running jobs.
INFO  16:23:34,689 ShellJobRunner - Starting: echo hello world
INFO  16:23:34,689 ShellJobRunner - Output written to
/Users/kshakir/src/Sting/Q-43031@bmef8-d8e-1.out
INFO  16:23:34,771 ShellJobRunner - Done: echo hello world
INFO  16:23:34,773 QGraph - Deleting intermediate files.
INFO  16:23:34,773 QCommandLine - Done
```

ExampleUnifiedGenotyper.scala

This example uses automatically generated Queue compatible wrappers for the GATK. See Pipelining the GATK using Queue for more info on authoring Queue support into walkers and using walkers in Queue.

The ExampleUnifiedGenotyper.scala for running the UnifiedGenotyper followed by VariantFiltration can be found in the examples folder.

To list the command line parameters, including the required parameters, run with `-help`.

```
java -jar dist/Queue.jar -S
public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/ExampleUnifiedGenotype
r.scala -help
```

The help output should appear similar to this:

```
INFO  10:26:08,491 QScriptManager - Compiling 1 QScript
INFO  10:26:11,926 QScriptManager - Compilation complete
-------------------------------------------------------
Program Name: org.broadinstitute.sting.queue.QCommandLine
-------------------------------------------------------
-------------------------------------------------------
usage: java -jar Queue.jar -S <script> [-run] [-jobRunner <job_runner>] [-bsub] [-status]
[-retry <retry_failed>]
       [-startFromScratch] [-keepIntermediates] [-statusTo <status_email_to>] [-statusFrom <
status_email_from>] [-dot
       <dot_graph>] [-expandedDot <expanded_dot_graph>] [-jobPrefix <job_name_prefix>]
[-jobProject <job_project>] [-jobQueue
       <job_queue>] [-jobPriority <job_priority>] [-memLimit <default_memory_limit>]
[-runDir <run_directory>] [-tempDir
       <temp_directory>] [-jobSGDir <job_scatter_gather_directory>] [-emailHost <
emailSmtpHost>] [-emailPort <emailSmtpPort>]
       [-emailTLS] [-emailSSL] [-emailUser <emailUsername>] [-emailPassFile <
emailPasswordFile>] [-emailPass <emailPassword>]
       [-l <logging_level>] [-log <log_to_file>] [-quiet] [-debug] [-h] -R <referencefile>
-I <bamfile> [-L <intervals>]
       [-filter <filternames>] [-filterExpression <filterexpressions>]

     -S,--script <script>                                                    QScript scala
file
     -run,--run_scripts                                                      Run QScripts.
 Without this flag set only
                                                                          performs a dry
run.
     -jobRunner,--job_runner <job_runner>                                    Use the
specified job runner to dispatch
```

```
                                                                    command line jobs
     -bsub,--bsub                                                   Equivalent to
-jobRunner Lsf706
     -status,--status                                               Get status of
jobs for the qscript
     -retry,--retry_failed <retry_failed>                           Retry the
specified number of times after a
                                                                    command fails.
Defaults to no retries.
     -startFromScratch,--start_from_scratch                         Runs all
command line functions even if the
                                                                    outputs were
previously output successfully.
     -keepIntermediates,--keep_intermediate_outputs                 After a
successful run keep the outputs of
                                                                    any Function
marked as intermediate.
     -statusTo,--status_email_to <status_email_to>                  Email address
to send emails to upon
                                                                    completion or on
error.
     -statusFrom,--status_email_from <status_email_from>            Email address
to send emails from upon
                                                                    completion or on

error.
     -dot,--dot_graph <dot_graph>                                   Outputs the
queue graph to a .dot file.  See:

http://en.wikipedia.org/wiki/DOT_language
     -expandedDot,--expanded_dot_graph <expanded_dot_graph>         Outputs the
queue graph of scatter gather to
                                                                    a .dot file.
Otherwise overwrites the
                                                                    dot_graph
     -jobPrefix,--job_name_prefix <job_name_prefix>                 Default name
prefix for compute farm jobs.
     -jobProject,--job_project <job_project>                        Default
project for compute farm jobs.
     -jobQueue,--job_queue <job_queue>                              Default queue
for compute farm jobs.
     -jobPriority,--job_priority <job_priority>                     Default
priority for jobs.
     -memLimit,--default_memory_limit <default_memory_limit>        Default
memory limit for jobs, in gigabytes.
     -runDir,--run_directory <run_directory>                        Root
directory to run functions from.
```

```
     -tempDir,--temp_directory <temp_directory>                                Temp
directory to pass to functions.
     -jobSGDir,--job_scatter_gather_directory <job_scatter_gather_directory>   Default
directory to place scatter gather
                                                                               output for
compute farm jobs.
     -emailHost,--emailSmtpHost <emailSmtpHost>                                Email SMTP
host. Defaults to localhost.
     -emailPort,--emailSmtpPort <emailSmtpPort>                                Email SMTP
port. Defaults to 465 for ssl,
                                                                               otherwise 25.
     -emailTLS,--emailUseTLS                                                   Email should
use TLS. Defaults to false.
     -emailSSL,--emailUseSSL                                                   Email should
use SSL. Defaults to false.
     -emailUser,--emailUsername <emailUsername>                                Email SMTP
username. Defaults to none.
     -emailPassFile,--emailPasswordFile <emailPasswordFile>                    Email SMTP
password file. Defaults to none.
     -emailPass,--emailPassword <emailPassword>                                Email SMTP
password. Defaults to none. Not
                                                                               secure! See
emailPassFile.
     -l,--logging_level <logging_level>                                        Set the
minimum level of logging, i.e.
                                                                               setting INFO
get's you INFO up to FATAL,
                                                                               setting ERROR
gets you ERROR and FATAL level
                                                                               logging.
     -log,--log_to_file <log_to_file>                                          Set the
logging location
     -quiet,--quiet_output_mode                                                Set the
logging to quiet mode, no output to
                                                                               stdout
     -debug,--debug_mode                                                       Set the
logging file string to include a lot
                                                                               of debugging
information (SLOW!)
     -h,--help                                                                 Generate this
help message

Arguments for ExampleUnifiedGenotyper:
     -R,--referencefile <referencefile>                          The reference file for the
bam files.
     -I,--bamfile <bamfile>                                      Bam file to genotype.
```

```
     -L,--intervals <intervals>                                    An optional file with a
list of intervals to proccess.
     -filter,--filternames <filternames>                           A optional list of filter
names.
     -filterExpression,--filterexpressions <filterexpressions>   An optional list of filter
expressions.


##### ERROR
------------------------------------------------------------------------------------------
##### ERROR stack trace
org.broadinstitute.sting.commandline.MissingArgumentException:
Argument with name '--bamfile' (-I) is missing.
Argument with name '--referencefile' (-R) is missing.
        at
org.broadinstitute.sting.commandline.ParsingEngine.validate(ParsingEngine.java:192)
        at
org.broadinstitute.sting.commandline.ParsingEngine.validate(ParsingEngine.java:172)
        at
org.broadinstitute.sting.commandline.CommandLineProgram.start(CommandLineProgram.java:199)
        at org.broadinstitute.sting.queue.QCommandLine$.main(QCommandLine.scala:57)
        at org.broadinstitute.sting.queue.QCommandLine.main(QCommandLine.scala)
##### ERROR
------------------------------------------------------------------------------------------
##### ERROR A GATK RUNTIME ERROR has occurred (version 1.0.5504):
##### ERROR
##### ERROR Please visit the wiki to see if this is a known problem
##### ERROR If not, please post the error, with stack trace, to the GATK forum
##### ERROR Visit our wiki for extensive documentation
http://www.broadinstitute.org/gsa/wiki
##### ERROR Visit our forum to view answers to commonly asked questions
http://getsatisfaction.com/gsa
##### ERROR
##### ERROR MESSAGE: Argument with name '--bamfile' (-I) is missing.
##### ERROR Argument with name '--referencefile' (-R) is missing.
##### ERROR
------------------------------------------------------------------------------------------
```

To dry run the pipeline:

```
java \
     -Djava.io.tmpdir=tmp \
     -jar dist/Queue.jar \
     -S
public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/ExampleUnifiedGenotype
r.scala \
```

```
    -R human_b36_both.fasta \
    -I pilot2_daughters.chr20.10k-11k.bam \
    -L chr20.interval_list \
    -filter StrandBias -filterExpression "SB>=0.10" \
    -filter AlleleBalance -filterExpression "AB>=0.75" \
    -filter QualByDepth -filterExpression "QD<5" \
    -filter HomopolymerRun -filterExpression "HRun>=4"
```

The dry run output should appear similar to this:

```
INFO  10:45:00,354 QScriptManager - Compiling 1 QScript
INFO  10:45:04,855 QScriptManager - Compilation complete
INFO  10:45:05,058 HelpFormatter - ---------------------------------------------------------
INFO  10:45:05,059 HelpFormatter - Program Name: org.broadinstitute.sting.queue.QCommandLine
INFO  10:45:05,059 HelpFormatter - Program Args: -S
public/scala/qscript/org/broadinstitute/sting/queue/qscripts/examples/ExampleUnifiedGenotype
r.scala -R human_b36_both.fasta -I pilot2_daughters.chr20.10k-11k.bam -L chr20.interval_list
-filter StrandBias -filterExpression SB>=0.10 -filter AlleleBalance -filterExpression AB>
=0.75 -filter QualByDepth -filterExpression QD<5 -filter HomopolymerRun -filterExpression
HRun>=4
INFO  10:45:05,059 HelpFormatter - Date/Time: 2011/03/24 10:45:05
INFO  10:45:05,059 HelpFormatter - ---------------------------------------------------------
INFO  10:45:05,059 HelpFormatter - ---------------------------------------------------------
INFO  10:45:05,061 QCommandLine - Scripting ExampleUnifiedGenotyper
INFO  10:45:05,150 QCommandLine - Added 4 functions
INFO  10:45:05,150 QGraph - Generating graph.
INFO  10:45:05,169 QGraph - Generating scatter gather jobs.
INFO  10:45:05,182 QGraph - Removing original jobs.
INFO  10:45:05,183 QGraph - Adding scatter gather jobs.
INFO  10:45:05,231 QGraph - Regenerating graph.
INFO  10:45:05,247 QGraph - -------
INFO  10:45:05,252 QGraph - Pending: IntervalScatterFunction
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-1/scatter.intervals
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-2/scatter.intervals
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-3/scatter.intervals
INFO  10:45:05,253 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/scatter/Q-60018@bmef8-d8e
-1.out
INFO  10:45:05,254 QGraph - -------
INFO  10:45:05,279 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T UnifiedGenotyper -I
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.bam -L
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-1/scatter.intervals
-R /Users/kshakir/src/Sting/human_b36_both.fasta -o
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-1/pilot2_daughters.c
```

```
hr20.10k-11k.unfiltered.vcf
INFO  10:45:05,279 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-1/Q-60018@bmef8-d8e-
1.out
INFO  10:45:05,279 QGraph - -------
INFO  10:45:05,283 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T UnifiedGenotyper -I
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.bam -L
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-2/scatter.intervals
-R /Users/kshakir/src/Sting/human_b36_both.fasta -o
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-2/pilot2_daughters.c
hr20.10k-11k.unfiltered.vcf
INFO  10:45:05,283 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-2/Q-60018@bmef8-d8e-
1.out
INFO  10:45:05,283 QGraph - -------
INFO  10:45:05,287 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T UnifiedGenotyper -I
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.bam -L
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-3/scatter.intervals
-R /Users/kshakir/src/Sting/human_b36_both.fasta -o
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-3/pilot2_daughters.c
hr20.10k-11k.unfiltered.vcf
INFO  10:45:05,287 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-3/Q-60018@bmef8-d8e-
1.out
INFO  10:45:05,288 QGraph - -------
INFO  10:45:05,288 QGraph - Pending: SimpleTextGatherFunction
/Users/kshakir/src/Sting/Q-60018@bmef8-d8e-1.out
INFO  10:45:05,288 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/gather-jobOutputFile/Q-60
018@bmef8-d8e-1.out
INFO  10:45:05,289 QGraph - -------
INFO  10:45:05,291 QGraph - Pending: java -Xmx1g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T CombineVariants -L
/Users/kshakir/src/Sting/chr20.interval_list -R
/Users/kshakir/src/Sting/human_b36_both.fasta -B:input0,VCF
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-1/pilot2_daughters.c
hr20.10k-11k.unfiltered.vcf -B:input1,VCF
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-2/pilot2_daughters.c
hr20.10k-11k.unfiltered.vcf -B:input2,VCF
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/temp-3/pilot2_daughters.c
hr20.10k-11k.unfiltered.vcf -o
```

```
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.unfiltered.vcf -priority
input0,input1,input2 -assumeIdenticalSamples
INFO  10:45:05,291 QGraph - Log:
/Users/kshakir/src/Sting/queueScatterGather/Q-60018@bmef8-d8e-1-sg/gather-out/Q-60018@bmef8-
d8e-1.out
INFO  10:45:05,292 QGraph - -------
INFO  10:45:05,296 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T VariantEval -L
/Users/kshakir/src/Sting/chr20.interval_list -R
/Users/kshakir/src/Sting/human_b36_both.fasta -B:eval,VCF
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.unfiltered.vcf -o
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.unfiltered.eval
INFO  10:45:05,296 QGraph - Log: /Users/kshakir/src/Sting/Q-60018@bmef8-d8e-2.out
INFO  10:45:05,296 QGraph - -------
INFO  10:45:05,299 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T VariantFiltration -L
/Users/kshakir/src/Sting/chr20.interval_list -R
/Users/kshakir/src/Sting/human_b36_both.fasta -B:vcf,VCF
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.unfiltered.vcf -o
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.filtered.vcf -filter SB>=0.10
-filter AB>=0.75 -filter QD<5 -filter HRun>=4 -filterName StrandBias -filterName
AlleleBalance -filterName QualByDepth -filterName HomopolymerRun
INFO  10:45:05,299 QGraph - Log: /Users/kshakir/src/Sting/Q-60018@bmef8-d8e-3.out
INFO  10:45:05,302 QGraph - -------
INFO  10:45:05,303 QGraph - Pending: java -Xmx2g
-Djava.io.tmpdir=/Users/kshakir/src/Sting/tmp -cp "/Users/kshakir/src/Sting/dist/Queue.jar"
org.broadinstitute.sting.gatk.CommandLineGATK -T VariantEval -L
/Users/kshakir/src/Sting/chr20.interval_list -R
/Users/kshakir/src/Sting/human_b36_both.fasta -B:eval,VCF
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.filtered.vcf -o
/Users/kshakir/src/Sting/pilot2_daughters.chr20.10k-11k.filtered.eval
INFO  10:45:05,303 QGraph - Log: /Users/kshakir/src/Sting/Q-60018@bmef8-d8e-4.out
INFO  10:45:05,304 QGraph - Dry run completed successfully!
INFO  10:45:05,304 QGraph - Re-run with "-run" to execute the functions.
INFO  10:45:05,304 QCommandLine - Done
```

8. Using traits to pass common values between QScripts to CommandLineFunctions

`QScript` files often create multiple `CommandLineFunctions` with similar arguments. Use various scala tricks such as inner classes, traits / mixins, etc. to reuse variables.

- A `self type` can be useful to distinguish between `this`. We use `qscript` as an alias for the QScript's `this` to distinguish from the `this` inside of inner classes or traits.
- A `trait mixin` can be used to reuse functionality. The trait below is designed to copy values from the

QScript and then is mixed into different instances of the functions.

See the following example:

```
class MyScript extends org.broadinstitute.sting.queue.QScript {
  // Create an alias 'qscript' for 'MyScript.this'
  qscript =>

  // This is a script argument
  @Argument(doc="message to display")
  var message: String = _

  // This is a script argument
  @Argument(doc="number of times to display")
  var count: Int = _

  trait ReusableArguments extends MyCommandLineFunction {
    // Whenever a function is created 'with' this trait, it will copy the message.
    this.commandLineMessage = qscript.message
  }

  abstract class MyCommandLineFunction extends CommandLineFunction {
     // This is a per command line argument
     @Argument(doc="message to display")
     var commandLineMessage: String = _
  }

  class MyEchoFunction extends MyCommandLineFunction {
     def commandLine = "echo " + commandLineMessage
  }

  class MyAlsoEchoFunction extends MyCommandLineFunction {
     def commandLine = "echo also " + commandLineMessage
  }

  def script = {
    for (i <- 1 to count) {
      val echo = new MyEchoFunction with ReusableArguments
      val alsoEcho = new MyAlsoEchoFunction with ReusableArguments
      add(echo, alsoEcho)
    }
  }
}
```

Queue with Grid Engine #1313

Last updated on 2012-10-18 15:39:32

1. Background

Thanks to contributions from the community, Queue contains a job runner compatible with Grid Engine 6.2u5.

As of July 2011 this is the currently known list of forked distributions of Sun's Grid Engine 6.2u5. As long as they are JDRMAA 1.0 source compatible with Grid Engine 6.2u5, the compiled Queue code should run against each of these distributions. However we have yet to receive confirmation that Queue works on any of these setups.

- Oracle Grid Engine 6.2u7
- Univa Grid Engine Core 8.0.0
- Univa Grid Engine 8.0.0
- Son of Grid Engine 8.0.0a
- Rocks 5.4 (includes a Roll for "SGE V62u5")
- Open Grid Scheduler 6.2u5p2

Our internal QScript integration tests run the same tests on both LSF 7.0.6 and a Grid Engine 6.2u5 cluster setup on older software released by Sun.

If you run into trouble, please let us know. If you would like to contribute additions or bug fixes please create a fork in our github repo where we can review and pull in the patch.

2. Running Queue with GridEngine

Try out the Hello World example with `-jobRunner GridEngine`.

```
java -Djava.io.tmpdir=tmp -jar dist/Queue.jar -S
public/scala/qscript/examples/HelloWorld.scala -jobRunner GridEngine -run
```

If all goes well Queue should dispatch the job to Grid Engine and wait until the status returns `RunningStatus.DONE` and "`hello world` should be echoed into the output file, possibly with other grid engine log messages.

See QFunction and Command Line Options for more info on Queue options.

3. Debugging issues with Queue and GridEngine

If you run into an error with Queue submitting jobs to GridEngine, first try submitting the HelloWorld example with `-memLimit 2`:

```
java -Djava.io.tmpdir=tmp -jar dist/Queue.jar -S
public/scala/qscript/examples/HelloWorld.scala -jobRunner GridEngine -run -memLimit 2
```

Then try the following GridEngine qsub commands. They are based on what Queue submits via the API when running the `HelloWorld.scala` example with and without memory reservations and limits:

```
qsub -w e -V -b y -N echo_hello_world \
     -o test.out -wd $PWD -j y echo hello world

qsub -w e -V -b y -N echo_hello_world \
     -o test.out -wd $PWD -j y \
     -l mem_free=2048M -l h_rss=2458M echo hello world
```

One other thing to check is if there is a memory limit on your cluster. For example try submitting jobs with up to 16G.

```
qsub -w e -V -b y -N echo_hello_world \
     -o test.out -wd $PWD -j y \
     -l mem_free=4096M -l h_rss=4915M echo hello world

qsub -w e -V -b y -N echo_hello_world \
     -o test.out -wd $PWD -j y \
     -l mem_free=8192M -l h_rss=9830M echo hello world

qsub -w e -V -b y -N echo_hello_world \
     -o test.out -wd $PWD -j y \
     -l mem_free=16384M -l h_rss=19960M echo hello world
```

If the above tests pass and GridEngine will still not dispatch jobs submitted by Queue please report the issue to our support forum.

Queue with IntelliJ IDEA #1309

Last updated on 2012-10-18 15:12:36

We have found it that Queue works best with IntelliJ IDEA Community Edition (free) or Ultimate Edition installed with the Scala Plugin enabled. Once you have downloaded IntelliJ IDEA, follow the instructions below to setup a Sting project with Queue and the Scala Plugin.

[[File:sting_project_libraries.png|300px|thumb|right|Project Libraries]]
[[File:sting_module_sources.png|300px|thumb|right|Module Sources]]
[[File:sting_module_dependencies.png|300px|thumb|right|Module Dependencies]]
[[File:sting_module_scala_facet.png|300px|thumb|right|Scala Facet]]

1. Build Queue on the Command Line

Build Queue from source from the command line with `ant queue`, so that: - The lib folder is initialized including the scala jars - The `queue-extensions` for the GATK are generated to the build folder

2. Add the scala plugin

- In IntelliJ, open the menu File ~ Settings

- Under the IDE Settings in the left navigation list select Plugins

- Click on the Available tab under plugins

- Scroll down in the list of available plugins and install the scala plugin

- If asked to retrieve dependencies, click No. The correct scala libraries and compiler are already available in the lib folder from when you built Queue from the command line

- Restart IntelliJ to load the scala plugin

3. Creating a new Sting Project including Queue

- Select the menu File... ~ New Project...

- On the first page of "New Project" select Create project from scratch Click Next >

- On the second page of "New Project" select Set the project Name: to Sting Set the Project files location: to the directory where you checked out the Sting git repository, for example /Users/jamie/src/Sting Uncheck Create Module Click Finish

- The "Project Structure" window should open. If not open it via the menu File ~ Project Structure

- Under the Project Settings in the left panel of "Project Structure" select Project Make sure that Project SDK is set to a build of 1.6 If the Project SDK only lists <No SDK> add a New ~ JSDK pointing to /System/Library/Frameworks/JavaVM.framework/Versions/1.6

- Under the Project Settings in the left panel of "Project Structure" select Libraries Click the plus (+) to create a new Project Library Set the Name: to Sting/lib Select Attach Jar Directories Select the path to lib folder under your SVN checkout

- Under the Project Settings in the left panel of "Project Structure" select Modules

- Click on the + box to add a new module

- On the first page of "Add Module" select Create module from scratch Click Next \>

- On the second page of "Add Module" select Set the module Name: to Sting Change the Content root to: <directory where you checked out the Sting SVN repository> Click Next \>

- On the third page Uncheck all of the other source directories only leaving the java/src directory checked Click Next \>

- On fourth page click Finish

- Back in the Project Structure window, under the Module 'Sting', on the Sources tab make sure the following folders are selected

- Source Folders (in blue): public/java/src public/scala/src private/java/src (Broad only) private/scala/src (Broad only) build/queue-extensions/src

- Test Source Folders (in green): public/java/test public/scala/test private/java/test (Broad only) private/scala/test (Broad only)

- In the Project Structure window, under the Module 'Sting', on the Module Dependencies tab

select Click on the button `Add...` Select the popup menu `Library...` Select the `Sting/lib` library Click `Add selected`

- Refresh the Project Structure window so that it becomes aware of the Scala library in `Sting/lib` Click the `OK` button Reopen Project Structure via the menu `File ~ Project Structure`

- In the second panel, click on the `Sting` module Click on the plus (+) button above the second panel module In the popup menu under `Facet` select `Scala` On the right under `Facet 'Scala'` set the `Compiler library:` to `Sting/lib` Click `OK`

4. Enable annotation processing

- Open the menu `File ~ Settings`

- Under `Project Settings [Sting]` in the left navigation list select `Compiler` then `Annotation Processors`

- Click to enable the checkbox `Enable annotation processing`

- Leave the radio button `obtain processors from the classpath` selected

- Click `OK`

5. Debugging Queue

Adding a Remote Configuration

[[File:queue_debug.png|300px|thumb|right|Queue Remote Debug]]

- In IntelliJ 10 open the menu `Run ~ Edit Configurations`.

- Click the gold `[+]` button at the upper left to open the `Add New Configuration` popup menu.

- Select `Remote` from the popup menu.

- With the new configuration selected on the left, change the configuration name from 'Unnamed' to something like 'Queue Remote Debug'.

- Set the `Host` to the hostname of your server, and the `Port` to an unused port. You can try the default port of 5005.

- From the `Use the following command line arguments for running remote JVM`, copy the argument string.

- On the server, paste / modify your command line to run with the previously copied text, for example `java -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005 Queue.jar -S myscript.scala ...`.

- If you would like the program to wait for you to attach the debugger before running, change `suspend=n` to `suspend=y`.

- Back in IntelliJ, click `OK` to save your changes.

Running with the Remote Configuration

- Ensure `Queue Remote Debug` is selected via the configuration drop down or `Run ~ Edit Configurations`.

- Set your breakpoints as you normally would in IntelliJ.

- Start your program by running the full java path (with the above -Xdebug -Xrunjdwp ...) on the server.
- In IntelliJ go to the `Run ~ Debug`.

6. Binding javadocs and source

From Stack overflow:

Add javadocs:

Point IntelliJ to http://download.oracle.com/javase/6/docs/api/.
Go to File -> Project Structure -> SDKs -> Apple 1.x -> DocumentationPaths, and the click specify URL.

Add sources:

In IntelliJ, open File -> Project Structure. Click on "SDKs" under "Platform Settings". Add the following path under the Sourcepath tab: /Library/Java/JavaVirtualMachines/1.6.0_29-b11-402.jdk/Contents/Home/src.jar!/src

Sampling and filtering reads #1323

Last updated on 2012-10-18 15:16:57

1. Introduction

Reads can be filtered out of traversals by either pileup size through one of our downsampling methods or by read property through our read filtering mechanism. Both techniques and described below.

2. Downsampling

Normal sequencing and alignment protocols can often yield pileups with vast numbers of reads aligned to a single section of the genome in otherwise well-behaved datasets. Because of the frequency of these 'speed bumps', the GATK now downsamples pileup data unless explicitly overridden.

Defaults

The GATK's default downsampler exhibits the following properties:

- The downsampler treats data from each sample independently, so that high coverage in one sample won't negatively impact calling in other samples.
- The downsampler attempts to downsample uniformly across the range spanned by the reads in the pileup.
- The downsampler's memory consumption is proportional to the sampled coverage depth rather than the full coverage depth.

By default, the downsampler is limited to 1000 reads per sample. This value can be adjusted either per-walker or per-run.

Customizing

From the command line:

- To disable the downsampler, specify `-dt NONE`.
- To change the default coverage per-sample, specify the desired coverage to the `-dcov` option.

To modify the walker's default behavior:

- Add the @Downsample interface to the top of your walker. Override the downsampling type by changing the `by=<value>`. Override the downsampling depth by changing the `toCoverage=<value>`.

Algorithm details

The downsampler algorithm is designed to maintain uniform coverage while preserving a low memory footprint in regions of especially deep data. Given an already established pileup, a single-base locus, and a pile of reads with an alignment start of single-base locus + 1, the outline of the algorithm is as follows:

For each sample:

- Select reads with the next alignment start.
- While the number of existing reads + the number of incoming reads is greater than the target sample size:

  Walk backward through each set of reads having the same alignment start. If the count of reads having the same alignment start is > 1, throw out one randomly selected read.
  - If we have n slots avaiable where n is >= 1, randomly select n of the incoming reads and add them to the pileup.
  - Otherwise, we have zero slots available. Choose the read from the existing pileup with the least alignment start. Throw it out and add one randomly selected read from the new pileup.

3. Read filtering

To selectively filter out reads before they reach your walker, implement one or multiple `net.sf.picard.filter.SamRecordFilter`, and attach it to your walker as follows:

```
@ReadFilters({Platform454Filter.class, ZeroMappingQualityReadFilter.class})
```

4. Command-line arguments for read filters

You can add command-line arguments for filters with the `@Argument` tag, just as with walkers. Here's an example of our new max read length filter:

```
public class MaxReadLengthFilter implements SamRecordFilter {
    @Argument(fullName = "maxReadLength", shortName = "maxRead", doc="Discard reads with
length greater than the specified value", required=false)
    private int maxReadLength;

    public boolean filterOut(SAMRecord read) { return read.getReadLength() > maxReadLength;
```

```
}
}
```

Adding this filter to the top of your walker using the `@ReadFilters` attribute will add a new required command-line argument, `maxReadLength`, which will filter reads > `maxReadLength` before your walker is called.

Note that when you specify a read filter, you need to strip the `Filter` part of its name off! E.g. in the example above, if you want to use `MaxReadLengthFilter`, you need to call it like this:

```
--read_filter MaxReadLength
```

5. Adding filters dynamically using command-line arguments

The `--read-filter` argument will allow you to apply whatever read filters you'd like to your dataset, before the reads reach your walker. To add the `MaxReadLength` filter above to `PrintReads`, you'd add the command line parameters:

```
--read_filter MaxReadLength --maxReadLength 76
```

You can add as many filters as you like by using multiple copies of the `--read_filter parameter`:

```
--read_filter MaxReadLength --maxReadLength 76 --read_filter ZeroMappingQualityRead
```

Scala resources #1897

Last updated on 2012-12-07 18:32:08

References for Scala development

The online course Functional Programming Principles in Scala taught by Martin Odersky, creator of Scala, and a Cheat Sheet for that course

Scala by Example (PDF) - also by Martin Odersky

First Steps to Scala

Programming Scala - O'Reilly Media

Scala School - Twitter

Scala Style Guide

A Concise Introduction To Scala

Scala Operator Cheat Sheet

A Tour of Scala

Stack Overflow

- Scala Punctuation (aka symbols, operators)
- What are all the uses of an underscore in Scala?

A Conversation with Martin Odersky

- The Origins of Scala
- The Goals of Scala's Design
- The Purpose of Scala's Type System
- The Point of Pattern Matching in Scala

Scala Collections for the Easily Bored

- A Tale of Two Flavors
- One at a Time
- All at Once

Seeing deletion spanning reads in LocusWalkers #1348

Last updated on 2012-10-18 15:24:35

1. Introduction

The `LocusTraversal` now supports passing walkers reads that have deletions spanning the current locus. This is useful in many situation where you want to calculate coverage, call variants and need to avoid calling variants where there are a lot of deletions, etc.

Currently, the system by default will not pass you deletion-spanning reads. In order to see them, you need to overload the function:

```
/**
 * (conceptual static) method that states whether you want to see reads piling up at a locus
 * that contain a deletion at the locus.
 *
 * ref:   ATCTGA
 * read1: ATCTGA
 * read2: AT--GA
 *
 * Normally, the locus iterator only returns a list of read1 at this locus at position 3,
but
 * if this function returns true, then the system will return (read1, read2) with offsets
 * of (3, -1).  The -1 offset indicates a deletion in the read.
 *
```

```
 * @return false if you don't want to see deletions, or true if you do
 */
public boolean includeReadsWithDeletionAtLoci() { return true; }
```

in your walker. Now you will start seeing deletion-spanning reads in your walker. These reads are flagged with offsets of -1, so that you can:

```
    for ( int i = 0; i < context.getReads().size(); i++ ) {
        SAMRecord read = context.getReads().get(i);
        int offset = context.getOffsets().get(i);

       if ( offset == -1 )
               nDeletionReads++;
        else
               nCleanReads++;
    }
```

There are also two convenience functions in `AlignmentContext` to extract subsets of the reads with and without spanning deletions:

```
/**
 * Returns only the reads in ac that do not contain spanning deletions of this locus
 *
 * @param ac
 * @return
 */
public static AlignmentContext withoutSpanningDeletions( AlignmentContext ac );

/**
 * Returns only the reads in ac that do contain spanning deletions of this locus
 *
 * @param ac
 * @return
 */
public static AlignmentContext withSpanningDeletions( AlignmentContext ac );
```

Tribble #1349

Last updated on 2012-10-18 15:23:58

1. Overview

The Tribble project was started as an effort to overhaul our reference-ordered data system; we had many different formats that were shoehorned into a common framework that didn't really work as intended. What we wanted was a common framework that allowed for searching of reference ordered data, regardless of the

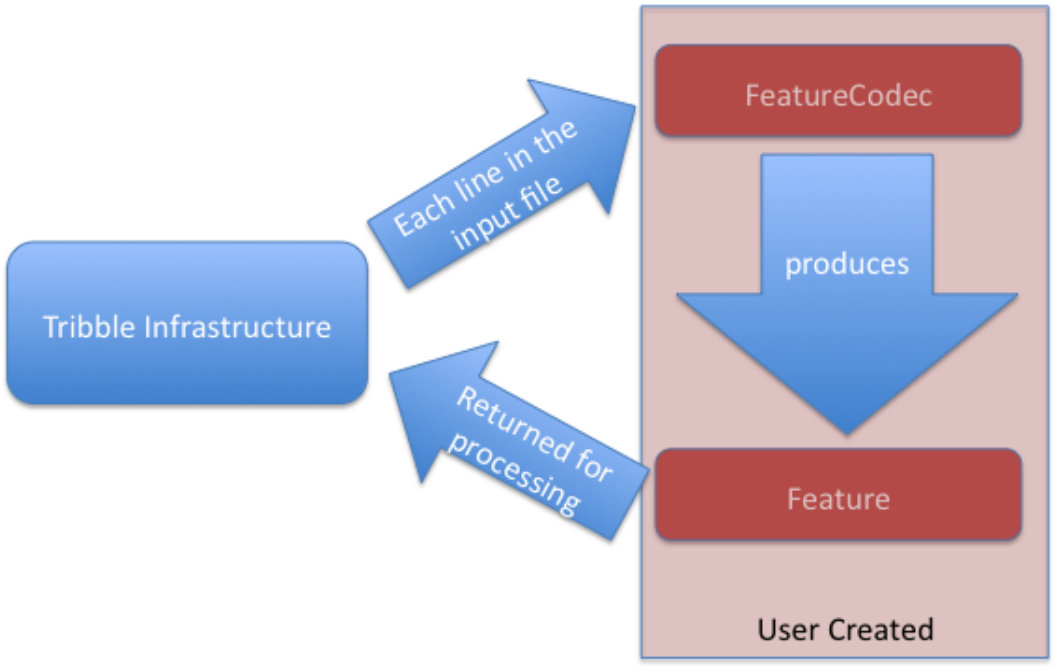
underlying type. Jim Robinson had developed indexing schemes for text-based files, which was incorporated into the Tribble library.

2. Architecture Overview

Tribble provides a lightweight interface and API for querying features and creating indexes from feature files, while allowing iteration over know feature files that we're unable to create indexes for. The main entry point for external users is the BasicFeatureReader class. It takes in a codec, an index file, and a file containing the features to be processed. With an instance of a `BasicFeatureReader`, you can query for features that span a specific location, or get an iterator over all the records in the file.

3. Developer Overview

For developers, there are two important classes to implement: the FeatureCodec, which decodes lines of text and produces features, and the feature class, which is your underlying record type.



For developers there are two classes that are important:

- **Feature**

This is the genomicly oriented feature that represents the underlying data in the input file. For instance in the VCF format, this is the variant call including quality information, the reference base, and the alternate base. The required information to implement a feature is the chromosome name, the start position (one based), and the stop position. The start and stop position represent a closed, one-based interval. I.e. the first base in chromosome one would be chr1:1-1.

- **FeatureCodec**

This class takes in a line of text (from an input source, whether it's a file, compressed file, or a http link), and produces the above feature.

To implement your new format into Tribble, you need to implement the two above classes (in an appropriately named subfolder in the Tribble check-out). The Feature object should know nothing about the file representation; it should represent the data as an in-memory object. The interface for a feature looks like:

```
public interface Feature {

    /**
     * Return the features reference sequence name, e.g chromosome or contig
     */
    public String getChr();

    /**
     * Return the start position in 1-based coordinates (first base is 1)
     */
    public int getStart();

    /**
     * Return the end position following 1-based fully closed conventions.  The length of a
feature is
     * end - start + 1;
     */
    public int getEnd();
}
```

And the interface for FeatureCodec:

```
/**
 * the base interface for classes that read in features.
 * @param <T> The feature type this codec reads
 */
public interface FeatureCodec<T extends Feature> {
    /**
     * Decode a line to obtain just its FeatureLoc for indexing -- contig, start, and stop.
     *
```

```
     * @param line the input line to decode
     * @return  Return the FeatureLoc encoded by the line, or null if the line does not
represent a feature (e.g. is
     * a comment)
     */
    public Feature decodeLoc(String line);

    /**
     * Decode a line as a Feature.
     *
     * @param line the input line to decode
     * @return  Return the Feature encoded by the line,  or null if the line does not
represent a feature (e.g. is
     * a comment)
     */
    public T decode(String line);

    /**
     * This function returns the object the codec generates.  This is allowed to be Feature
in the case where
     * conditionally different types are generated.  Be as specific as you can though.
     *
     * This function is used by reflections based tools, so we can know the underlying type
     *
     * @return the feature type this codec generates.
     */
    public Class<T> getFeatureType();


    /**  Read and return the header, or null if there is no header.
     *
     * @return header object
     */
    public Object readHeader(LineReader reader);
}
```

4. Supported Formats

The following formats are supported in Tribble:

- VCF Format
- DbSNP Format
- BED Format
- GATK Interval Format

5. Updating the Tribble library

Updating the revision of Tribble on the system is a relatively straightforward task if the following steps are taken.

- Make sure that you've checked your changes into Tribble; unversioned changes will be problematic, so you should always check in so that you have a unique version number to identify your release.

- Once you've checked-in Tribble, make sure to `svn update`, and then run `svnversion`. This will give you a version number which you can use to name your release. Let's say it was 82. **If it contains an M (i.e. 82M) this means your version isn't clean (you have modifications that are not checked in), don't proceed`.

- from the Tribble main directory, run `ant clean`, then `ant` (make sure it runs successfully), and `ant test` (also make sure it completes successfully).

- copy `dist/tribble-0.1.jar` (or whatever the internal Tribble version currently is) to your checkout of the GATK, as the file `./settings/repository/org.broad/tribble-<svnversion>.jar`.

- Copy the current XML file to the new name, i.e. from the base GATK trunk directory:

cp ./settings/repository/org.broad/tribble-.xml ./settings/repository/org.broad/tribble-.xml

- Edit the `./settings/repository/org.broad/tribble-<svnversion>.xml` with the new correct version number and release date (here we rev 81 to 82).

This involves changing:

```
<ivy-module version="1.0">
    <info organisation="org.broad" module="tribble" revision="81" status="integration"
publication="20100526124200" />
</ivy-module>
```

To:

```
<ivy-module version="1.0">
    <info organisation="org.broad" module="tribble" revision="82" status="integration"
publication="20100528123456" />
</ivy-module>
```

Notice the change to the revision number and the publication date.

Notice the change to the revision number and the publication date.

- Remove the old files `svn remove ./settings/repository/org.broad/tribble-<current_svnversion>.*`

- Add the new files `svn add ./settings/repository/org.broad/tribble-<new_svnversion>.*`

- Make sure you're using the new libraries to build: remove your ant cache: `rm -r ~/.ant/cache`.

- Run an `ant clean`, and then make sure to test the build with `ant integrationtest` and `ant test`.

- Any check-in from the base SVN directory will now rev the Tribble version.

Using DiffEngine to summarize differences between structured data files #1299

Last updated on 2012-10-18 15:43:46

1. What is DiffEngine?

DiffEngine is a summarizing difference engine that allows you to compare two structured files -- such as BAMs and VCFs -- to find what are the differences between them. This is primarily useful in regression testing or optimization, where you want to ensure that the differences are those that you expect and not any others.

2. The summarized differences

The GATK contains a summarizing difference engine called DiffEngine that compares hierarchical data structures to emit:

- A list of specific differences between the two data structures. This is similar to saying the value in field A in record 1 in file F differs from the value in field A in record 1 in file G.
- A summarized list of differences ordered by frequency of the difference. This output is similar to saying field A differed in 50 records between files F and G.

3. The DiffObjects walker

The GATK contains a private walker called DiffObjects that allows you access to the DiffEngine capabilities on the command line. Simply provide the walker with the master and test files and it will emit summarized differences for you.

4. Understanding the output

The DiffEngine system compares to two hierarchical data structures for specific differences in the values of named nodes. Suppose I have two trees:

```
Tree1=(A=1 B=(C=2 D=3))
Tree2=(A=1 B=(C=3 D=3 E=4))
Tree3=(A=1 B=(C=4 D=3 E=4))
```

where every node in the tree is named, or is a raw value (here all leaf values are integers). The DiffEngine traverses these data structures by name, identifies equivalent nodes by fully qualified names (`Tree1.A` is distinct from `Tree2.A`, and determines where their values are equal (`Tree1.A=1`, `Tree2.A=1`, so they are).

These itemized differences are listed as:

```
Tree1.B.C=2 != Tree2.B.C=3
Tree1.B.C=2 != Tree3.B.C=4
Tree2.B.C=3 != Tree3.B.C=4
Tree1.B.E=MISSING != Tree2.B.E=4
```

This conceptually very similar to the output of the unix command line tool `diff`. What's nice about DiffEngine though is that it computes similarity among the itemized differences and displays the count of differences names in the system. In the above example, the field `C` is not equal three times, while the missing `E` in `Tree1` occurs only once. So the summary is:

```
*.B.C : 3
*.B.E : 1
```

where the * operator indicates that any named field matches. This output is sorted by counts, and provides an immediate picture of the commonly occurring differences between the files.

Below is a detailed example of two VCF fields that differ because of a bug in the `AC`, `AF`, and `AN` counting routines, detected by the `integrationtest` integration (more below). You can see that in the although there are many specific instances of these differences between the two files, the summarized differences provide an immediate picture that the `AC`, `AF`, and `AN` fields are the major causes of the differences.

```
[testng] path                                                              count
[testng] *.*.*.AC                                                          6
[testng] *.*.*.AF                                                          6
[testng] *.*.*.AN                                                          6
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000000.AC  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000000.AF  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000000.AN  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000117.AC  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000117.AF  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000117.AN  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000211.AC  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000211.AF  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000211.AN  1
[testng] 64b991fd3850f83614518f7d71f0532f.integrationtest.20:10000598.AC  1
```

5. Integration tests

The DiffEngine codebase that supports these calculations is integrated into the `integrationtest` framework, so that when a test fails the system automatically summarizes the differences between the master MD5 file and the failing MD5 file, if it is an understood type. When failing you will see in the integration test logs not only the basic information, but the detailed DiffEngine output.

For example, in the output below I broke the GATK BAQ calculation and the integration test DiffEngine clearly identifies that all of the records differ in their `BQ` tag value in the two BAM files:

```
/humgen/1kg/reference/human_b36_both.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.allTechs.bam
-o
/var/folders/Us/UsMJ3xRrFVyuDXWkUos1xkC43FQ/-Tmp-/walktest.tmp_param.05785205687740257584.tm
p -L 1:10,000,000-10,100,000 -baq RECALCULATE -et NO_ET
   [testng] WARN  22:59:22,875 TextFormattingUtils - Unable to load help text.  Help output
```

```
will be sparse.
   [testng] WARN  22:59:22,875 TextFormattingUtils - Unable to load help text.  Help output
will be sparse.
   [testng] ##### MD5 file is up to date:
integrationtests/e5147656858fc4a5f470177b94b1fc1b.integrationtest
   [testng] Checking MD5 for
/var/folders/Us/UsMJ3xRrFVyuDXWkUos1xkC43FQ/-Tmp-/walktest.tmp_param.05785205687740257584.tm
p [calculated=e5147656858fc4a5f470177b94b1fc1b, expected=4ac691bde1ba1301a59857694fda6ae2]
   [testng] ##### Test testPrintReadsRecalBAQ is going fail #####
   [testng] ##### Path to expected   file (MD5=4ac691bde1ba1301a59857694fda6ae2):
integrationtests/4ac691bde1ba1301a59857694fda6ae2.integrationtest
   [testng] ##### Path to calculated file (MD5=e5147656858fc4a5f470177b94b1fc1b):
integrationtests/e5147656858fc4a5f470177b94b1fc1b.integrationtest
   [testng] ##### Diff command: diff
integrationtests/4ac691bde1ba1301a59857694fda6ae2.integrationtest
integrationtests/e5147656858fc4a5f470177b94b1fc1b.integrationtest
   [testng] ##:GATKReport.v0.1 diffences : Summarized differences between the master and
test files.
   [testng] See
http://www.broadinstitute.org/gsa/wiki/index.php/DiffObjectsWalker_and_SummarizedDifferences
for more information
   [testng] Difference
        NumberOfOccurrences
   [testng] *.*.*.BQ
        895
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:2:266:272:361.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:5:245:474:254.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:5:255:178:160.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:6:158:682:495.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:6:195:591:884.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:7:165:236:848.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:7:191:223:910.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAE_0002_FC205W7AAXX:7:286:279:434.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAF_0002_FC205Y7AAXX:2:106:516:354.BQ  1
   [testng]
4ac691bde1ba1301a59857694fda6ae2.integrationtest.-XAF_0002_FC205Y7AAXX:3:102:580:518.BQ  1
   [testng]
```

```
   [testng] Note that the above list is not comprehensive.  At most 20 lines of output, and
10 specific differences will be listed.  Please use -T DiffObjects -R
public/testdata/exampleFASTA.fasta -m
integrationtests/4ac691bde1ba1301a59857694fda6ae2.integrationtest -t
integrationtests/e5147656858fc4a5f470177b94b1fc1b.integrationtest to explore the differences
more freely
```

6. Adding your own DiffableObjects to the system

The system dynamically finds all classes that implement the following simple interface:

```
public interface DiffableReader {
    @Ensures("result != null")
    /**
     * Return the name of this DiffableReader type.  For example, the VCF reader returns
'VCF' and the
     * bam reader 'BAM'
     */
    public String getName();

    @Ensures("result != null")
    @Requires("file != null")
    /**
     * Read up to maxElementsToRead DiffElements from file, and return them.
     */
    public DiffElement readFromFile(File file, int maxElementsToRead);

    /**
     * Return true if the file can be read into DiffElement objects with this reader. This
should
     * be uniquely true/false for all readers, as the system will use the first reader that
can read the
     * file.  This routine should never throw an exception.  The VCF reader, for example,
looks at the
     * first line of the file for the ##format=VCF4.1 header, and the BAM reader for the
BAM_MAGIC value
     * @param file
     * @return
     */
    @Requires("file != null")
    public boolean canRead(File file);
```

See the VCF and BAMDiffableReaders for example implementations. If you extend this to a new object types both the DiffObjects walker and the `integrationtest` framework will automatically work with your new file type.

Writing GATKdocs for your walkers #1324

Last updated on 2012-10-18 15:35:49

The GATKDocs are what we call "Technical Documentation" in the Guide section of this website. The HTML pages are generated automatically at build time from specific blocks of documentation in the source code.

The best place to look for example documentation for a GATK walker is GATKDocsExample walker in `org.broadinstitute.sting.gatk.examples`. This is available here.

Below is the reproduction of that file from August 11, 2011:

```
/**
 * [Short one sentence description of this walker]
 *
 * <p>
 * [Functionality of this walker]
 * </p>
 *
 * <h2>Input</h2>
 * <p>
 * [Input description]
 * </p>
 *
 * <h2>Output</h2>
 * <p>
 * [Output description]
 * </p>
 *
 * <h2>Examples</h2>
 * PRE-TAG
 *    java
 *      -jar GenomeAnalysisTK.jar
 *      -T $WalkerName
 * PRE-TAG
 *
 * @category Walker Category
 * @author Your Name
 * @since Date created
 */
public class GATKDocsExample extends RodWalker<Integer, Integer> {
    /**
     * Put detailed documentation about the argument here.  No need to duplicate the summary
information
     * in doc annotation field, as that will be added before this text in the documentation
page.
     *
```

```
     * Notes:
     * <ul>
     *     <li>This field can contain HTML as a normal javadoc</li>
     *     <li>Don't include information about the default value, as gatkdocs adds this
automatically</li>
     *     <li>Try your best to describe in detail the behavior of the argument, as
ultimately confusing
     *          docs here will just result in user posts on the forum</li>
     * </ul>
     */
    @Argument(fullName="full", shortName="short", doc="Brief summary of argument [~ 80
characters of text]", required=false)
    private boolean myWalkerArgument = false;

    public Integer map(RefMetaDataTracker tracker, ReferenceContext ref, AlignmentContext
context) { return 0; }
    public Integer reduceInit() { return 0; }
    public Integer reduce(Integer value, Integer sum) { return value + sum; }
    public void onTraversalDone(Integer result) { }
}
```

Writing and working with reference metadata classes #1350

Last updated on 2012-10-18 15:23:11

Brief introduction to reference metadata (RMDs)

Note that the `-B` flag referred to below is deprecated; these docs need to be updated

The GATK allows you to process arbitrary numbers of reference metadata (RMD) files inside of walkers (previously we called this reference ordered data, or ROD). Common RMDs are things like dbSNP, VCF call files, and refseq annotations. The only real constraints on RMD files is that:

- They must contain information necessary to provide contig and position data for each element to the GATK engine so it knows with what loci to associate the RMD element.
- The file must be sorted with regard to the reference fasta file so that data can be accessed sequentially by the engine.
- The file must have a Tribble RMD parsing class associated with the file type so that elements in the RMD file can be parsed by the engine.

Inside of the GATK the RMD system has the concept of RMD tracks, which associate an arbitrary string name with the data in the associated RMD file. For example, the `VariantEval` module uses the named track `eval` to get calls for evaluation, and `dbsnp` as the track containing the database of known variants.

How do I get reference metadata files into my walker?

RMD files are extremely easy to get into the GATK using the `-B` syntax:

```
java -jar GenomeAnalysisTK.jar -R Homo_sapiens_assembly18.fasta -T PrintRODs -B:variant,VCF calls.vcf
```

In this example, the GATK will attempt to parse the file `calls.vcf` using the VCF parser and bind the VCF data to the RMD track named `variant`.

In general, you can provide as many RMD bindings to the GATK as you like:

```
java -jar GenomeAnalysisTK.jar -R Homo_sapiens_assembly18.fasta -T PrintRODs -B:calls1,VCF calls1.vcf -B:calls2,VCF calls2.vcf
```

Works just as well. Some modules may require specifically named RMD tracks -- like `eval` above -- and some are happy to just assess all RMD tracks of a certain class and work with those -- like `VariantsToVCF`.

1. Directly getting access to a single named track

In this snippet from `SNPDensityWalker`, we grab the `eval` track as a `VariantContext` object, only for the variants that are of type SNP:

```
public Pair<VariantContext, GenomeLoc> map(RefMetaDataTracker tracker, ReferenceContext ref, AlignmentContext context) {
    VariantContext vc = tracker.getVariantContext(ref, "eval", EnumSet.of(VariantContext.Type.SNP), context.getLocation(), false);
}
```

2. Grabbing anything that's convertable to a VariantContext

From `VariantsToVCF` we call the helper function `tracker.getVariantContexts` to look at all of the RMDs and convert what it can to `VariantContext` objects.

```
Allele refAllele = new Allele(Character.toString(ref.getBase()), true);
Collection<VariantContext> contexts = tracker.getVariantContexts(INPUT_RMD_NAME, ALLOWED_VARIANT_CONTEXT_TYPES, context.getLocation(), refAllele, true, false);
```

3. Looking at all of the RMDs

Here's a totally general code snippet from `PileupWalker.java`. This code, as you can see, iterates over all of the GATKFeature objects in the reference ordered data, converting each RMD to a string and capturing these strings in a list. It finally grabs the dbSNP binding specifically for a more detailed string conversion, and then binds them all up in a single string for display along with the read pileup.

private String getReferenceOrderedData(RefMetaDataTracker tracker) { ArrayList rodStrings = new ArrayList(); for (GATKFeature datum : tracker.getAllRods()) { if (datum != null && ! (datum.getUnderlyingObject() instanceof DbSNPFeature)) {

rodStrings.add(((ReferenceOrderedDatum)datum.getUnderlyingObject()).toSimpleString()); // TODO: Aaron: this line still survives, try to remove it } } String rodString = Utils.join(", ", rodStrings);

```
        DbSNPFeature dbsnp = tracker.lookup(DbSNPHelper.STANDARD_DBSNP_TRACK_NAME,
DbSNPFeature.class);

        if ( dbsnp != null)
            rodString += DbSNPHelper.toMediumString(dbsnp);

        if ( !rodString.equals("") )
            rodString = "[ROD: " + rodString + "]";

        return rodString;
}
```

How do I write my own RMD types?

Tracks of reference metadata are loaded using the Tribble infrastructure. Tracks are loaded using the feature codec and underlying type information. See the Tribble documentation for more information.

Tribble codecs that are in the classpath are automatically found; the GATK discovers all classes that implement the `FeatureCodec` class. Name resolution occurs using the `-B` type parameter, i.e. if the user specified:

```
-B:calls1,VCF calls1.vcf
```

The GATK looks for a `FeatureCodec` called `VCFCodec.java` to decode the record type. Alternately, if the user specified:

```
-B:calls1,MYAwesomeFormat calls1.maft
```

THe GATK would look for a codec called `MYAwesomeFormatCodec.java`. This look-up is not case sensitive, i.e. it will resolve `MyAwEsOmEfOrMaT` as well, though why you would want to write something so painfully ugly to read is beyond us.

Writing unit / regression tests for QScripts #1353

Last updated on 2012-10-18 15:18:30

In addition to testing walkers individually, you may want to also run integration tests for your QScript pipelines.

1. Brief comparison to the Walker integration tests

- Pipeline tests should use the standard location for testing data.
- Pipeline tests use the same test dependencies.
- Pipeline tests which generate MD5 results will have the results stored in the MD5 database].

- Pipeline tests, like QScripts, are written in Scala.
- Pipeline tests dry-run under the `ant` target `pipelinetest` and run under `pipelinetestrun`.
- Pipeline tests class names must end in `PipelineTest` to run under the `ant` target.
- Pipeline tests should instantiate a `PipelineTestSpec` and then run it via `PipelineTest.exec()`.

2. PipelineTestSpec

When building up a pipeline test spec specify the following variables for your test.

Variable	Type	Description
args	String	The arguments to pass to the Queue test, ex: -S scalaqscriptexamplesHelloWorld.scala
jobQueue	String	Job Queue to run the test. Default is null which means use hour.
fileMD5s	Map[Path, MD5]	Expected MD5 results for each file path.
expectedException	classOf[Exception]	Expected exception from the test.

3. Example PipelineTest

The following example runs the `ExampleCountLoci` QScript on a small bam and verifies that the MD5 result is as expected.

It is checked into the Sting repository under `scala/test/org/broadinstitute/sting/queue/pipeline/examples/ExampleCountLociPipelineTest.scala`

```
package org.broadinstitute.sting.queue.pipeline.examples

import org.testng.annotations.Test
import org.broadinstitute.sting.queue.pipeline.{PipelineTest, PipelineTestSpec}
import org.broadinstitute.sting.BaseTest

class ExampleCountLociPipelineTest {
  @Test
  def testCountLoci {
    val testOut = "count.out"
    val spec = new PipelineTestSpec
    spec.name = "countloci"
    spec.args = Array(
      " -S scala/qscript/examples/ExampleCountLoci.scala",
      " -R " + BaseTest.hg18Reference,
      " -I " + BaseTest.validationDataLocation + "small_bam_for_countloci.bam",
      " -o " + testOut).mkString
    spec.fileMD5s += testOut -> "67823e4722495eb10a5e4c42c267b3a6"
    PipelineTest.executeTest(spec)
  }
```

```
}
```

3. Running Pipeline Tests

Dry Run

To test if the script is at least compiling with your arguments run `ant pipelinetest` specifying the name of your class to `-Dsingle`:

```
ant pipelinetest -Dsingle=ExampleCountLociPipelineTest
```

Sample output:

```
   [testng] ------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour
   [testng]   => countloci PASSED DRY RUN
   [testng] PASSED: testCountLoci
```

Run

As of July 2011 the pipeline tests run against LSF 7.0.6 and Grid Engine 6.2u5. To include these two packages in your environment use the hidden dotkit `.combined_LSF_SGE`.

```
reuse .combined_LSF_SGE
```

Once you are satisfied that the dry run has completed without error, to actually run the pipeline test run `ant pipelinetestrun`.

```
ant pipelinetestrun -Dsingle=ExampleCountLociPipelineTest
```

Sample output:

```
   [testng] ------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour -run
   [testng] ##### MD5 file is up to date:
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] Checking MD5 for pipelinetests/countloci/run/count.out
```

```
[calculated=67823e4722495eb10a5e4c42c267b3a6, expected=67823e4722495eb10a5e4c42c267b3a6]
   [testng]   => countloci PASSED
   [testng] PASSED: testCountLoci
```

Generating initial MD5s

If you don't know the MD5s yet you can run the command yourself on the command line and then MD5s the outputs yourself, or you can set the MD5s in your test to "" and run the pipeline.

When the MD5s are blank as in:

```
spec.fileMD5s += testOut -> ""
```

You run:

```
ant pipelinetest -Dsingle=ExampleCountLociPipelineTest -Dpipeline.run=run
```

And the output will look like:

```
   [testng] ------------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour -run
   [testng] ##### MD5 file is up to date:
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] PARAMETERIZATION[countloci]: file pipelinetests/countloci/run/count.out has md5
= 67823e4722495eb10a5e4c42c267b3a6, stated expectation is , equal? = false
   [testng]   => countloci PASSED
   [testng] PASSED: testCountLoci
```

Checking MD5s

When a pipeline test fails due to an MD5 mismatch you can use the MD5 database to diff the results.

```
   [testng] ------------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour -run
   [testng] ##### Updating MD5 file:
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] Checking MD5 for pipelinetests/countloci/run/count.out
```

```
[calculated=67823e4722495eb10a5e4c42c267b3a6, expected=67823e4722495eb10a5e0000deadbeef]
   [testng] ##### Test countloci is going fail #####
   [testng] ##### Path to expected   file (MD5=67823e4722495eb10a5e0000deadbeef):
integrationtests/67823e4722495eb10a5e0000deadbeef.integrationtest
   [testng] ##### Path to calculated file (MD5=67823e4722495eb10a5e4c42c267b3a6):
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] ##### Diff command: diff
integrationtests/67823e4722495eb10a5e0000deadbeef.integrationtest
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] FAILED: testCountLoci
   [testng] java.lang.AssertionError: 1 of 1 MD5s did not match.
```

If you need to examine a number of MD5s which may have changed you can briefly shut off MD5 mismatch failures by setting `parameterize = true`.

```
spec.parameterize = true
spec.fileMD5s += testOut -> "67823e4722495eb10a5e4c42c267b3a6"
```

For this run:

```
ant pipelinetest -Dsingle=ExampleCountLociPipelineTest -Dpipeline.run=run
```

If there's a match the output will resemble:

```
   [testng] ------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour -run
   [testng] ##### MD5 file is up to date:
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] PARAMETERIZATION[countloci]: file pipelinetests/countloci/run/count.out has md5
= 67823e4722495eb10a5e4c42c267b3a6, stated expectation is 67823e4722495eb10a5e4c42c267b3a6,
equal? = true
   [testng]   => countloci PASSED
   [testng] PASSED: testCountLoci
```

While for a mismatch it will look like this:

```
   [testng] ------------------------------------------------------------------------
   [testng] Executing test countloci with Queue arguments: -S
scala/qscript/examples/ExampleCountLoci.scala -R
/seq/references/Homo_sapiens_assembly18/v0/Homo_sapiens_assembly18.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/small_bam_for_countloci.bam -o count.out
```

```
-bsub -l WARN -tempDir pipelinetests/countloci/temp/ -runDir pipelinetests/countloci/run/
-jobQueue hour -run
   [testng] ##### MD5 file is up to date:
integrationtests/67823e4722495eb10a5e4c42c267b3a6.integrationtest
   [testng] PARAMETERIZATION[countloci]: file pipelinetests/countloci/run/count.out has md5
= 67823e4722495eb10a5e4c42c267b3a6, stated expectation is 67823e4722495eb10a5e0000deadbeef,
equal? = false
   [testng]   => countloci PASSED
   [testng] PASSED: testCountLoci
```

Writing unit tests for walkers #1339

Last updated on 2012-10-18 15:28:56

1. Testing core walkers is critical

Most GATK walkers are really too complex to easily test using the standard unit test framework. It's just not feasible to make artificial read piles and then extrapolate from simple tests passing whether the system as a whole is working correctly. However, we need some way to determine whether changes to the core of the GATK are altering the expected output of complex walkers like BaseRecalibrator or SingleSampleGenotyper. In additional to correctness, we want to make sure that the performance of key walkers isn't degrading over time, so that calling snps, cleaning indels, etc., isn't slowly creeping down over time. Since we are now using a bamboo server to automatically build and run unit tests (as well as measure their runtimes) we want to put as many good walker tests into the test framework so we capture performance metrics over time.

2. The WalkerTest framework

To make this testing process easier, we've created a `WalkerTest` framework that lets you invoke the GATK using command-line GATK commands in the `JUnit` system and test for changes in your output files by comparing the current ant build results to previous run via an MD5 sum. It's a bit coarse grain, but it will work to ensure that changes to key walkers are detected quickly by the system, and authors can either update the expected MD5s or go track down bugs.

The system is fairly straightforward to use. Ultimately we will end up with `JUnit` style tests in the unit testing structure. In the piece of code below, we have a piece of code that checks the MD5 of the SingleSampleGenotyper's GELI text output at LOD 3 and LOD 10.

```
package org.broadinstitute.sting.gatk.walkers.genotyper;

import org.broadinstitute.sting.WalkerTest;
import org.junit.Test;

import java.util.HashMap;
import java.util.Map;
import java.util.Arrays;
```

```
public class SingleSampleGenotyperTest extends WalkerTest {
    @Test
    public void testLOD() {
        HashMap<Double, String> e = new HashMap<Double, String>();
        e.put( 10.0, "e4c51dca6f1fa999f4399b7412829534" );
        e.put( 3.0, "d804c24d49669235e3660e92e664ba1a" );

        for ( Map.Entry<Double, String> entry : e.entrySet() ) {
            WalkerTest.WalkerTestSpec spec = new WalkerTest.WalkerTestSpec(
                   "-T SingleSampleGenotyper -R /broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam -varout %s
--variant_output_format GELI -L 1:10,000,000-11,000,000 -m EMPIRICAL -lod " +
entry.getKey(), 1,
                    Arrays.asList(entry.getValue()));
            executeTest("testLOD", spec);
        }
    }
}
```

The fundamental piece here is to inherit from `WalkerTest`. This gives you access to the `executeTest()` function that consumes a `WalkerTestSpec`:

```
    public WalkerTestSpec(String args, int nOutputFiles, List<String> md5s)
```

The `WalkerTestSpec` takes regular, command-line style GATK arguments describing what you want to run, the number of output files the walker will generate, and your expected MD5s for each of these output files. The args string can contain `%s String.format` specifications, and for each of the `nOutputFiles`, the `executeTest()` function will (1) generate a `tmp` file for output and (2) call `String.format` on your args to fill in the tmp output files in your arguments string. For example, in the above argument string `varout` is followed by `%s`, so our single SingleSampleGenotyper output is the variant output file.

3. Example output

When you add a `walkerTest` inherited unit test to the GATK, and then `build test`, you'll see output that looks like:

```
[junit] WARN  13:29:50,068 WalkerTest -
------------------------------------------------------------------------------
[junit] WARN  13:29:50,068 WalkerTest -
------------------------------------------------------------------------------
[junit] WARN  13:29:50,069 WalkerTest - Executing test testLOD with GATK arguments: -T
SingleSampleGenotyper -R /broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam -varout
/tmp/walktest.tmp_param.05524470250256847817.tmp --variant_output_format GELI -L
1:10,000,000-11,000,000 -m EMPIRICAL -lod 3.0
[junit]
```

```
[junit] WARN  13:29:50,069 WalkerTest - Executing test testLOD with GATK arguments: -T
SingleSampleGenotyper -R /broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam -varout
/tmp/walktest.tmp_param.05524470250256847817.tmp --variant_output_format GELI -L
1:10,000,000-11,000,000 -m EMPIRICAL -lod 3.0
[junit]
[junit] WARN  13:30:39,407 WalkerTest - Checking MD5 for
/tmp/walktest.tmp_param.05524470250256847817.tmp
[calculated=d804c24d49669235e3660e92e664ba1a, expected=d804c24d49669235e3660e92e664ba1a]
[junit] WARN  13:30:39,407 WalkerTest - Checking MD5 for
/tmp/walktest.tmp_param.05524470250256847817.tmp
[calculated=d804c24d49669235e3660e92e664ba1a, expected=d804c24d49669235e3660e92e664ba1a]
[junit] WARN  13:30:39,408 WalkerTest -   => testLOD PASSED
[junit] WARN  13:30:39,408 WalkerTest -   => testLOD PASSED
[junit] WARN  13:30:39,409 WalkerTest -
-------------------------------------------------------------------------------
[junit] WARN  13:30:39,409 WalkerTest -
-------------------------------------------------------------------------------
[junit] WARN  13:30:39,409 WalkerTest - Executing test testLOD with GATK arguments: -T
SingleSampleGenotyper -R /broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam -varout
/tmp/walktest.tmp_param.03852477489430798188.tmp --variant_output_format GELI -L
1:10,000,000-11,000,000 -m EMPIRICAL -lod 10.0
[junit]
[junit] WARN  13:30:39,409 WalkerTest - Executing test testLOD with GATK arguments: -T
SingleSampleGenotyper -R /broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam -varout
/tmp/walktest.tmp_param.03852477489430798188.tmp --variant_output_format GELI -L
1:10,000,000-11,000,000 -m EMPIRICAL -lod 10.0
[junit]
[junit] WARN  13:31:30,213 WalkerTest - Checking MD5 for
/tmp/walktest.tmp_param.03852477489430798188.tmp
[calculated=e4c51dca6f1fa999f4399b7412829534, expected=e4c51dca6f1fa999f4399b7412829534]
[junit] WARN  13:31:30,213 WalkerTest - Checking MD5 for
/tmp/walktest.tmp_param.03852477489430798188.tmp
[calculated=e4c51dca6f1fa999f4399b7412829534, expected=e4c51dca6f1fa999f4399b7412829534]
[junit] WARN  13:31:30,213 WalkerTest -   => testLOD PASSED
[junit] WARN  13:31:30,213 WalkerTest -   => testLOD PASSED
[junit] WARN  13:31:30,214 SingleSampleGenotyperTest -
[junit] WARN  13:31:30,214 SingleSampleGenotyperTest -
```

4. Recommended location for GATK testing data

We keep all of the permenant GATK testing data in:

```
/humgen/gsa-scr1/GATK_Data/Validation_Data/
```

A good set of data to use for walker testing is the CEU daughter data from 1000 Genomes:

```
gsa2 ~/dev/GenomeAnalysisTK/trunk > ls -ltr
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_1*.bam
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_1*.calls
     -rw-rw-r--+ 1 depristo wga  51M 2009-09-03 07:56
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam
     -rw-rw-r--+ 1 depristo wga 185K 2009-09-04 13:21
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.lod5.variants.
geli.calls
     -rw-rw-r--+ 1 depristo wga 164M 2009-09-04 13:22
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.lod5.genotypes
.geli.calls
     -rw-rw-r--+ 1 depristo wga  24M 2009-09-04 15:00
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SOLID.bam
     -rw-rw-r--+ 1 depristo wga  12M 2009-09-04 15:01
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.454.bam
     -rw-r--r--+ 1 depristo wga  91M 2009-09-04 15:02
/humgen/gsa-scr1/GATK_Data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.allTechs.bam
```

5. Test dependencies

The tests depend on a variety of input files, that are generally constrained to three mount points on the internal Broad network:

```
*/seq/
*/humgen/1kg/
*/humgen/gsa-hpprojects/GATK/Data/Validation_Data/
```

To run the unit and integration tests you'll have to have access to these files. They may have different mount points on your machine (say, if you're running remotely over the VPN and have mounted the directories on your own machine).

6. MD5 database and comparing MD5 results

Every file that generates an MD5 sum as part of the WalkerTest framework will be copied to `<MD5>.integrationtest` in the `integrationtests` subdirectory of the GATK trunk. This MD5 database of results enables you to easily examine the results of an integration test as well as compare the results of a test before/after a code change. For example, below is an example test for the UnifiedGenotyper that, due to a code change, where the output VCF differs from the VCF with the expected MD5 value in the test code itself. The test provides provides the path to the two results files as well as a diff command to compare expected to the observed MD5:

```
[junit] ------------------------------------------------------------------------------
[junit] Executing test testParameter[-genotype] with GATK arguments: -T UnifiedGenotyper -R
```

```
/broad/1KG/reference/human_b36_both.fasta -I
/humgen/gsa-hpprojects/GATK/data/Validation_Data/NA12878.1kg.p2.chr1_10mb_11_mb.SLX.bam
-varout /tmp/walktest.tmp_param.05997727998894311741.tmp -L 1:10,000,000-10,010,000
-genotype
[junit] ##### MD5 file is up to date:
integrationtests/ab20d4953b13c3fc3060d12c7c6fe29d.integrationtest
[junit] Checking MD5 for /tmp/walktest.tmp_param.05997727998894311741.tmp
[calculated=ab20d4953b13c3fc3060d12c7c6fe29d, expected=0ac7ab893a3f550cb1b8c34f28baedf6]
[junit] ##### Test testParameter[-genotype] is going fail #####
[junit] ##### Path to expected   file (MD5=0ac7ab893a3f550cb1b8c34f28baedf6):
integrationtests/0ac7ab893a3f550cb1b8c34f28baedf6.integrationtest
[junit] ##### Path to calculated file (MD5=ab20d4953b13c3fc3060d12c7c6fe29d):
integrationtests/ab20d4953b13c3fc3060d12c7c6fe29d.integrationtest
[junit] ##### Diff command: diff
integrationtests/0ac7ab893a3f550cb1b8c34f28baedf6.integrationtest
integrationtests/ab20d4953b13c3fc3060d12c7c6fe29d.integrationtest
```

Examining the diff we see a few lines that have changed the DP count in the new code

```
> diff integrationtests/0ac7ab893a3f550cb1b8c34f28baedf6.integrationtest
integrationtests/ab20d4953b13c3fc3060d12c7c6fe29d.integrationtest  | head
385,387c385,387
< 1     10000345        .       A       .       106.54  .
AN=2;DP=33;Dels=0.00;MQ=89.17;MQ0=0;SB=-10.00   GT:DP:GL:GQ
0/0:25:-0.09,-7.57,-75.74:74.78
< 1     10000346        .       A       .       103.75  .
AN=2;DP=31;Dels=0.00;MQ=88.85;MQ0=0;SB=-10.00   GT:DP:GL:GQ
0/0:24:-0.07,-7.27,-76.00:71.99
< 1     10000347        .       A       .       109.79  .
AN=2;DP=31;Dels=0.00;MQ=88.85;MQ0=0;SB=-10.00   GT:DP:GL:GQ
0/0:26:-0.05,-7.85,-84.74:78.04
---
> 1     10000345        .       A       .       106.54  .
AN=2;DP=32;Dels=0.00;MQ=89.50;MQ0=0;SB=-10.00   GT:DP:GL:GQ
0/0:25:-0.09,-7.57,-75.74:74.78
> 1     10000346        .       A       .       103.75  .
AN=2;DP=30;Dels=0.00;MQ=89.18;MQ0=0;SB=-10.00   GT:DP:GL:GQ
0/0:24:-0.07,-7.27,-76.00:71.99
> 1     10000347        .       A       .       109.79  .
AN=2;DP=30;Dels=0.00;MQ=89.18;MQ0=0;SB=-10.00   GT:DP:GL:GQ     0/0:26:-0.05,-7.85,-84.74:78
```

Whether this is the expected change is up to you to decide, but the system makes it as easy as possible to see the consequences of your code change.

7. Testing for Exceptions

The walker test framework supports an additional syntax for ensuring that a particular java Exception is thrown when a walker executes using a simple alternate version of the `WalkerSpec` object. Rather than specifying the MD5 of the result, you can provide a single subclass of `Exception.class` and the testing framework will ensure that when the walker runs an instance (class or subclass) of your expected exception is thrown. The system also flags if no exception is thrown.

For example, the following code tests that the GATK can detect and error out when incompatible VCF and FASTA files are given:

```
@Test public void fail8() { executeTest("hg18lex-v-b36", test(lexHG18, callsB36)); }

private WalkerTest.WalkerTestSpec test(String ref, String vcf) {
    return new WalkerTest.WalkerTestSpec("-T VariantsToTable -M 10 -B:two,vcf "
            + vcf + " -F POS,CHROM -R "
            + ref +  " -o %s",
            1, UserException.IncompatibleSequenceDictionaries.class);

}
```

During the integration test this looks like:

```
[junit] Executing test hg18lex-v-b36 with GATK arguments: -T VariantsToTable -M 10
-B:two,vcf /humgen/gsa-hpprojects/GATK/data/Validation_Data/lowpass.N3.chr1.raw.vcf -F
POS,CHROM -R /humgen/gsa-hpprojects/GATK/data/Validation_Data/lexFasta/lex.hg18.fasta -o
/tmp/walktest.tmp_param.05541601616101756852.tmp -l WARN -et NO_ET
[junit]    [junit] Wanted exception class
org.broadinstitute.sting.utils.exceptions.UserException$IncompatibleSequenceDictionaries,
saw class
org.broadinstitute.sting.utils.exceptions.UserException$IncompatibleSequenceDictionaries
[junit]   => hg18lex-v-b36 PASSED
```

8. Miscellaneous information

- Please do not put any extremely long tests in the regular `ant build test` target. We are currently splitting the system into fast and slow tests so that unit tests can be run in \< 3 minutes while saving a test target for long-running regression tests. More information on that will be posted.

- An expected MG5 string of `""` means don't check for equality between the calculated and expected MD5s. Useful if you are just writing a new test and don't know the true output.

- Overload `parameterize() { return true; }` if you want the system to just run your calculations, not throw an error if your MD5s don't match, across all tests

- If your tests all of a sudden stop giving equality MD5s, you can just (1) look at the `.tmp` output files directly or (2) grab the printed GATK command-line options and explore what is happening.

- You can always run a GATK walker on the command line and then run md5sum on its output files to

obtain, outside of the testing framework, the MD5 expected results.

- Don't worry about the duplication of lines in the output ; it's just an annoyance of having two global loggers. Eventually we'll bug fix this away.

Writing walkers #1302

Last updated on 2013-03-06 19:55:08

1. Introduction

The core concept behind GATK tools is the walker, a class that implements the three core operations: **filtering**, **mapping**, and **reducing**.

- **filter** Reduces the size of the dataset by applying a predicate.
- **map** Applies a function to each individual element in a dataset, effectively mapping it to a new element.
- **reduce** Inductively combines the elements of a list. The base case is supplied by the `reduceInit()` function, and the inductive step is performed by the `reduce()` function.

Users of the GATK will provide a walker to run their analyses. The engine will produce a result by first filtering the dataset, running a map operation, and finally reducing the map operation to a single result.

2. Creating a Walker

To be usable by the GATK, the walker must satisfy the following properties:

- It must subclass one of the basic walkers in the `org.broadinstitute.sting.gatk.walkers` package, usually ReadWalker or LociWalker.

- Locus walkers present all the reads, reference bases, and reference-ordered data that overlap a single base in the reference. Locus walkers are best used for analyses that look at each locus independently, such as genotyping.
- Read walkers present only one read at a time, as well as the reference bases and reference-ordered data that overlap that read.
- Besides read walkers and locus walkers, the GATK features several other data access patterns, described here.

- The compiled class or jar must be on the current classpath. The Java classpath can be controlled using either the `$CLASSPATH` environment variable or the JVM's `-cp` option.

3. Examples

The best way to get started with the GATK is to explore the walkers we've written. Here are the best walkers to look at when getting started:

- CountLoci

It is the simplest locus walker in our codebase. It counts the number of loci walked over in a single run of the GATK.

`$STING_HOME/java/src/org/broadinstitute/sting/gatk/walkers/qc/CountLociWalker.java`

- CountReads

It is the simplest read walker in our codebase. It counts the number of reads walked over in a single run of the GATK.

`$STING_HOME/java/src/org/broadinstitute/sting/gatk/walkers/qc/CountReadsWalker.java`

- GATKPaperGenotyper

This is a more sophisticated example, taken from our recent paper in Genome Research (and using our ReadBackedPileup to select and filter reads). It is an extremely basic Bayesian genotyper that demonstrates how to output data to a stream and execute simple base operations.

`$STING_HOME/java/src/org/broadinstitute/sting/gatk/examples/papergenotyper/GATKPaperGenotyper.java`

Please note that the walker above is NOT the UnifiedGenotyper. While conceptually similar to the UnifiedGenotyper, the GATKPaperGenotyper uses a much simpler calling model for increased clarity and readability.

4. External walkers and the 'external' directory

The GATK can absorb external walkers placed in a directory of your choosing. By default, that directory is called 'external' and is relative to the Sting git root directory (for example, `~/src/Sting/external`). However, you can choose to place that directory anywhere on the filesystem and specify its complete path using the ant `external.dir` property.

```
ant -Dexternal.dir=~/src/external
```

The GATK will check each directory under the external directory (but not the external directory itself!) for small build scripts. These build scripts must contain at least a `compile` target that compiles your walker and places the resulting class file into the GATK's class file output directory. The following is a sample compile target:

```
<target name="compile" depends="init">
    <javac srcdir="." destdir="${build.dir}" classpath="${gatk.classpath}" />
</target>
```

As a convenience, the `build.dir` ant property will be predefined to be the GATK's class file output directory and the `gatk.classpath` property will be predefined to be the GATK's core classpath. Once this structure is defined, any invocation of the ant build scripts will build the contents of the external directory as well as the GATK itself.

Writing walkers in Scala #1354

Last updated on 2012-10-18 15:17:47

1. Install scala somewhere

At the Broad, we typically put it somewhere like this:

```
/home/radon01/depristo/work/local/scala-2.7.5.final
```

Next, create a symlink from this directory to `trunk/scala/installation`:

```
ln -s /home/radon01/depristo/work/local/scala-2.7.5.final trunk/scala/installation
```

2. Setting up your path

Right now the only way to get scala walkers into the GATK is by explicitly setting your CLASSPATH in your `.my.cshrc` file:

```
setenv CLASSPATH
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/FourBaseRecaller.jar:/humgen/gsa-s
cr1/depristo/dev/GenomeAnalysisTK/trunk/dist/GenomeAnalysisTK.jar:/humgen/gsa-scr1/depristo/
dev/GenomeAnalysisTK/trunk/dist/Playground.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisT
K/trunk/dist/StingUtils.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/bcel-5
.2.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/colt-1.2.0.jar:/humgen/gsa-
scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/google-collections-0.9.jar:/humgen/gsa-scr1/de
pristo/dev/GenomeAnalysisTK/trunk/dist/javassist-3.7.ga.jar:/humgen/gsa-scr1/depristo/dev/Ge
nomeAnalysisTK/trunk/dist/junit-4.4.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk
/dist/log4j-1.2.15.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/picard-1.02
.63.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/picard-private-875.jar:/hu
mgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/reflections-0.9.2.jar:/humgen/gsa-scr
1/depristo/dev/GenomeAnalysisTK/trunk/dist/sam-1.01.63.jar:/humgen/gsa-scr1/depristo/dev/Gen
omeAnalysisTK/trunk/dist/simple-xml-2.0.4.jar:/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK
/trunk/dist/GATKScala.jar:/humgen/gsa-scr1/depristo/local/scala-2.7.5.final/lib/scala-librar
y.jar
```

Really this needs to be manually updated whenever any of the libraries are updated. If you see this error:

```
Caused by: java.lang.RuntimeException: java.util.zip.ZipException: error in opening zip file
        at org.reflections.util.VirtualFile.iterable(VirtualFile.java:79)
        at org.reflections.util.VirtualFile$5.transform(VirtualFile.java:169)
        at org.reflections.util.VirtualFile$5.transform(VirtualFile.java:167)
        at org.reflections.util.FluentIterable$3.transform(FluentIterable.java:43)
        at org.reflections.util.FluentIterable$3.transform(FluentIterable.java:41)
        at
org.reflections.util.FluentIterable$ForkIterator.computeNext(FluentIterable.java:81)
        at
```

```
com.google.common.collect.AbstractIterator.tryToComputeNext(AbstractIterator.java:132)
        at com.google.common.collect.AbstractIterator.hasNext(AbstractIterator.java:127)
        at
org.reflections.util.FluentIterable$FilterIterator.computeNext(FluentIterable.java:102)
        at
com.google.common.collect.AbstractIterator.tryToComputeNext(AbstractIterator.java:132)
        at com.google.common.collect.AbstractIterator.hasNext(AbstractIterator.java:127)
        at
org.reflections.util.FluentIterable$TransformIterator.computeNext(FluentIterable.java:124)
        at
com.google.common.collect.AbstractIterator.tryToComputeNext(AbstractIterator.java:132)
        at com.google.common.collect.AbstractIterator.hasNext(AbstractIterator.java:127)
        at org.reflections.Reflections.scan(Reflections.java:69)
        at org.reflections.Reflections.<init>(Reflections.java:47)
        at org.broadinstitute.sting.utils.PackageUtils.<clinit>(PackageUtils.java:23)
```

It's because the libraries aren't updated. Basically just do an `ls` of your `trunk/dist` directory after the GATK has been build, make this your classpath as above, and tack on:

```
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/GATKScala.jar:/humgen/gsa-scr1/dep
risto/local/scala-2.7.5.final/lib/scala-library.jar
```

A command that almost works (but you'll need to replace the spaces with colons) is:

```
#setenv CLASSPATH $CLASSPATH `ls
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/*.jar`
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/GATKScala.jar:/humgen/gsa-scr1/dep
risto/local/scala-2.7.5.final/lib/scala-library.jar
```

3. Building scala code

All of the Scala source code lives in `scala/src`, which you build using `ant scala`

There are already some example Scala walkers in `scala/src`, so doing a standard checkout, installing scala, settting up your environment, should allow you to run something like:

```
gsa2 ~/dev/GenomeAnalysisTK/trunk > ant scala
Buildfile: build.xml

init.scala:

scala:
     [echo] Sting: Compiling scala!
   [scalac] Compiling 2 source files to
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/scala/classes
   [scalac] warning: there were deprecation warnings; re-run with -deprecation for details
```

```
   [scalac] one warning found
   [scalac] Compile suceeded with 1 warning; see the compiler output for details.
   [delete] Deleting:
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/GATKScala.jar
      [jar] Building jar:
/humgen/gsa-scr1/depristo/dev/GenomeAnalysisTK/trunk/dist/GATKScala.jar
```

4. Invoking a scala walker

Until we can include Scala walkers along with the main GATK jar (avoiding the classpath issue too) you have to invoke your scala walkers using this syntax:

```
java -Xmx2048m org.broadinstitute.sting.gatk.CommandLineGATK -T
BaseTransitionTableCalculator -R /broad/1KG/reference/human_b36_both.fasta -I
/broad/1KG/DCC_merged/freeze5/NA12878.pilot2.SLX.bam -l INFO -L 1:1-100
```

Here, the `BaseTransitionTableCalculator` walker is written in Scala and being loaded into the system by the GATK walker manager. Otherwise everything looks like a normal GATK module.

Third-Party Tools

Other teams have developed their own tools to work on top of the GATK framework. This section lists several of these software packages as well as links to documentation and contact information for their respective authors.

Please keep in mind that since this is not our software, we make no guarantees as to their use and cannot provide any support.

GenomeSTRiP

Bob Handsaker, Broad Institute

Genome STRiP (Genome STRucture In Populations) is a suite of tools for discovering and genotyping structural variations using sequencing data. The methods are designed to detect shared variation using data from multiple individuals, but can also process single genomes.

Please see the GenomeSTRiP website for more information:
http://www.broadinstitute.org/software/genomestrip/

You can ask questions and report problems about GenomeSTRiP in this category of the GATK forum:
http://gatkforums.broadinstitute.org/categories/genomestrip

MuTect

Kristian Cibulskis, Broad Institute

MuTect is a method developed at the Broad Institute for the reliable and accurate identification of somatic point mutations in next generation sequencing data of cancer genomes.

Please see the MuTect website for more information:
http://www.broadinstitute.org/cancer/cga/mutect

You can ask questions and report problems about MuTect in this category of the GATK forum:
http://gatkforums.broadinstitute.org/categories/mutect

XHMM

Menachem Fromer, Mt Sinai School of Medicine

The XHMM (eXome-Hidden Markov Model) C++ software suite calls copy number variation (CNV) from next-generation sequencing projects, where exome capture was used (or targeted sequencing, more generally). Specifically, XHMM uses principal component analysis (PCA) normalization and a hidden Markov model (HMM) to detect and genotype copy number variation (CNV) from normalized read-depth data from targeted sequencing experiments.

Please see the XHMM website for more information:
http://atgu.mgh.harvard.edu/xhmm/

You can ask questions and report problems about XHMM in this category of the GATK forum:
http://gatkforums.broadinstitute.org/categories/xhmm

See also the XHMM Google Group

Version History

These articles track the changes made in each major and minor version release (for example, 2.2). **Version highlights** are meant to give an overview of the key improvements and explain their significance. **Release notes* list all major changes as well as minor changes and bug fixes. At this time, we do not provide release notes for subversion changes (for example, 2.2-12).

Version highlights for GATK release 2.7 #3175

Posted on 2013-08-30 13:55:05

Yay, August is over! Goodbye steamy hot days, hello mild temperatures and beautiful leaf-peeping season. We hope you all had a great summer (in the Northern hemisphere at least) and caught a bit of a vacation. For our part, we've been chained to our desks the whole time!

Well, not really, but we've got a feature-rich release for you nonetheless. Lots of new things; not all of them fully mature, so heed the caveats on the experimental features! We've also made some key improvements to VQSR that we're very excited about, some bug fixes to various tools of course, and a new way to boost calling performance. Full list in the release notes as usual, and highlights below.

Estimating the confidence of reference calls

When UnifiedGenotyper and HaplotypeCaller emit variant calls, they tell you how confident you can be that the variants are real. But how do you know how confident to be that the rest are reference, i.e. non-variant? It's actually a pretty hard problemâ€¦ and this is our answer:

- For HaplotypeCaller, weâ€™ve developed a full-on reference model that produces reference confidence scores. To use it, you need to enable the `--emitRefConfidence` mode. This mode is a little bit complicated so be sure you read the method article before you try to use it.

- For UnifiedGenotyper, we donâ€™t have a completely fleshed-out model, but weâ€™ve added the `-allSitePLs` argument which, in combination with the `EMIT_ALL_SITES` output mode, will enable calculation of PLs for all sites, including reference. This will give a measure of reference confidence and a measure of which alt alleles are more plausible (if any). Note that this only works with the SNP calling model. Again, this is not as good or as complete as the reference model in HaplotypeCaller, so we urge you to use HaplotypeCaller for this unless you really need to use UnifiedGenotyper.

These are two highly experimental features; they work in our tests, but your mileage may vary, so please examine your results carefully. We welcome your feedback!

Modelling PCR errors that cause indel artifacts

A common problem in calling indels is that you get false positives that are associated with PCR slippage around short tandem repeats (especially homopolymers). Until we can all switch to PCR-free amplification, we're stuck with this issue. So we thought it would be nice to be able to model this type of error and mitigate its impact on our indel calls. The new `--pcr_indel_model` argument allows the HaplotypeCaller to use a new feature called the PCR indel model to weed out false positive indels more or less aggressively depending on how much you care about sensitivity vs. specificity.

This feature too is highly experimental, so play with it at your own risk. And stay tuned, because we've already got some ideas on how to improve it further.

VariantRecalibrator gets an oil change and free tire rotation

Variant recalibration is one of the most challenging parts of the Best Practices workflow, and not just for users! We've been wrestling with some of its internal machinery to produce better, more consistent modeling results, especially with call sets that are on the lower end of the size scale.

One of the breakthroughs we made was separating the parameters for the positive and negative training models. You know (or should know) that the VariantRecalibrator builds two separate models: one to model what "good" variants (i.e. true positives) look like (the positive model), and one to model what "bad" variants (i.e. false positives) look like (the negative model). Until now, we applied parameters the same way to both, but we've now realized that it makes more sense to treat them differently.

Because of how relative amounts of good and bad variants tend to scale differently with call set size, we also realized it was a bad idea to have the selection of bad variants be based on a percentage (as it has been until now) and instead switched it to a hard number. You can change this setting with the `--numBadVariants` argument, which replaces the now-deprecated `--percentBadVariants` argument.

Finally, we also found that the order of annotations matters. Now, instead of applying the annotation dimensions to the training model in the order that they were specified at the command line, VariantRecalibrator first reorders them based on their standard deviation. This stabilizes the training model and produces much more consistent results.

New arguments and tools for finer control of data

Some of you have been clamoring for more flexibility in handling individual BAM files and samples without losing the convenience of processing them in batches. In response, we've added the following:

- For general GATK use, the `-sample_rename_mapping_file` engine argument allows you to rename samples on-the-fly at runtime. It takes a file that maps bam files to sample names. Note that this does require that your BAM files contain single samples only, although multiple read groups are allowed.

- For variant calling, the `-onlyEmitSamples` argument allows you to tell the UnifiedGenotyper to only emit calls for specific samples among a cohort that you're calling in multisample mode, without emitting the calls for the rest of the cohort. Keep in mind however that the calculations will still be made on the entire cohort, and the annotation values emitted for those calls will reflect that.

- For VQSR, the `--excludeFiltered` flag tells the ApplyRecalibration tool not to emit sites that are filtered out by recalibration (i.e. do not write them to file).

And some of you went ahead and added the features you wanted yourselves!

- Yossi Farjoun contributed a patch to enable allele-biased downsampling with different per-sample values for the HaplotypeCaller, emulating the equivalent functionality that was already available in the UnifiedGenotyper.

- Louis Bergelson contributed a new read filter, LibraryReadFilter, which allows you to use only reads from a specific library in your analysis. This is the opposite (and somewhat more specific) functionality compared to the existing engine argument, --read_group_black_list , which allows you to exclude read groups based on specific tags (including but not limited to LB).

Better diagnostics when things go wrong

We have a new diagnostic tool, QualifyMissingIntervals, that allows you to collect metrics such as GC content, mapping quality etc. for a list of intervals of interest. This is something you'd typically want to use if you found (through other tools) that you're missing calls in certain intervals, and you want to find out what's going wrong in those regions.

FPGA support for the pairHMM model in UG and HC

Finally, those of you who have access to more sophisticated computing platforms, heads up! Version 2.7 comes with a version of the PairHMM algorithm (aka the bit that takes forever to run in HaplotypeCaller) that is optimized for running on FPGA chips. Credit goes to the fine folks at Convey Computer and Green Mountain Computing Systems who teamed up to develop this optimized version of the PairHMM, with a little help from our very own Tech Dev team. We're told further optimizations may be in store; in the meantime, they're seeing up to 300-fold speedups of HaplotypeCaller runs on Convey's platform. Not bad!

Version highlights for GATK release 2.6 #2846

Last updated on 2013-06-24 16:47:36

It's finally summer here in New England -- time for cave-dwelling developers to hit the beach and do the lobster dance (those of us who don't tan well anyway). We leave you with a new version of the GATK that includes a new(ish) plotting tool, some more performance improvements to the callers, a lot of feature tweaks and quite a

few bug fixes. Be sure to check out the full list in the 2.6 Release Notes.

Highlights are below as usual, enjoy. There's one thing that we need to point out with particular emphasis: we have moved to Java 7, so you may need to update your system's Java version. Full explanation at the end of this document because it's a little long, but be sure to read it.

New(ish) plotting tool for Base Recalibration results

GATK old-timers may remember a tool called AnalyzeCovariates, which was part of the BQSR process in 1.x versions, many moons ago. Well, we've resurrected it to take over the plotting functionality of the BaseRecalibrator, to make it easier and faster to plot and compare the results of base recalibration. This also prevents issues with plot generation in scatter-gather mode. We'll update our docs on the BQSR workflow in the next few days, but in the meantime you can find full details of how to use this tool here.

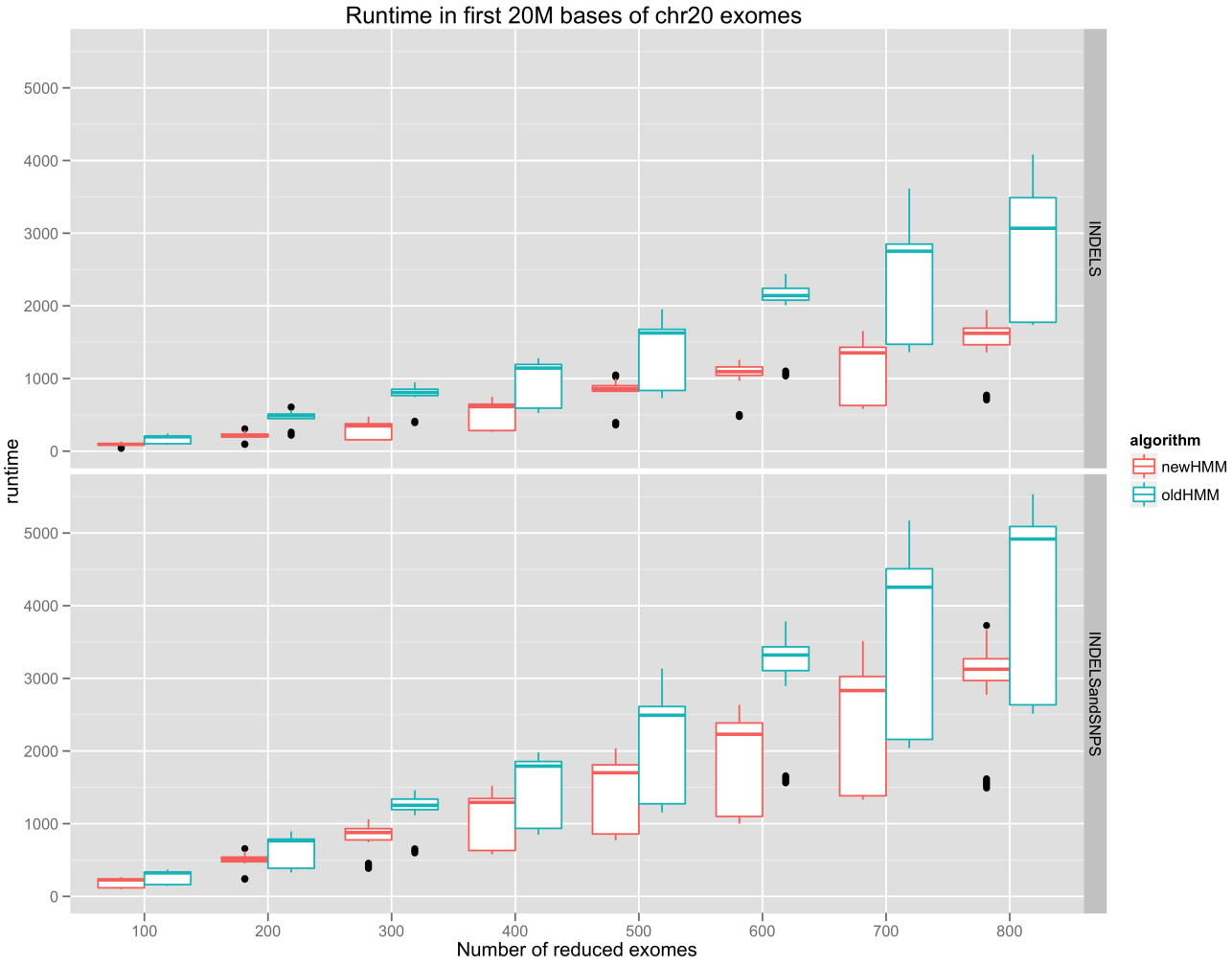
HaplotypeCaller now so sensitive, it cries at the movies

We know you don't want to miss a single true variant, so for this release, we've put a lot of effort into making the HaplotypeCaller more sensitive. And it's paying off: in our tests, the HaplotypeCaller is now more sensitive than the UnifiedGenotyper for calling both SNPs and indels when run over whole genome datasets.

[graph to illustrate, coming soon]

UnifiedGenotyper: not out of the race yet

You might think all our focus is on improving the HaplotypeCaller these days; you would be wrong. The UnifiedGenotyper is still essential for calling large numbers of samples together, for dealing with exotic ploidies, and for calling pooled samples. So we've given it a turbo boost that makes it go twice as fast for calling indels on multiple samples.



The key change here is the updated Hidden Markov Model used by the UG. You can see on the graph that as the number of exomes being called jointly increases, the new HMM keeps runtimes down significantly compared to the old HMM.

Version tracking in the VCF header

Donâ€™t you hate it when you go back to a VCF you generated some months ago, and you have no idea which version of GATK you used at the time? (And yes, versions matter. Sometimes a lot.) We sure do, so we added a function to add the GATK version number in the header of the VCFs generated by GATK.

Migration to Java 7

Speaking of software versions... As you probably know, the GATK runs on Java -- specifically, until now, version 6 of the Runtime Environment (which translates to version 1.6 if you ask `java -version` at the command prompt). But the Java language has been evolving under our feet; version 7 has been out and stable for some time now, and version 8 is on the horizon. We were happy as clams with Java 6â€¦ but now, newer computers with recent OS versions ship with Java 7, and on MacOS X once you update the system it is difficult to go back to using Java 6. And since Java 7 is not fully backwards compatible, people have been running into version problems.

So, we have made the difficult but necessary decision to follow the tide, and migrate the GATK to Java 7. Starting with this release, GATK will now require Java 7 to run. If you try to run with Java 6, you will probably get an error like this:

```
Exception in thread "main" java.lang.UnsupportedClassVersionError:
org/broadinstitute/sting/gatk/CommandLineGATK : Unsupported major.minor version 51.0
```

If you're not sure what version of Java you are currently using, you can find out very easily by typing the following command:

```
java -version
```

which should return something like this:

```
java version "1.7.0_17"
Java(TM) SE Runtime Environment (build 1.7.0_17-b02)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
```

If not, you'll need to update your java version. If you have any difficulty doing this, please donâ€™t ask us in the forum -- youâ€™ll get much better, faster help if you ask your local IT department.

Version highlights for GATK release 2.5 #2619

Last updated on 2013-09-16 20:43:47

Overview

This is going to be a short one, folks. The 2.5 release is pretty much all about bug fixes, with a couple of exceptions that we'll cover below.

Bug fixes

Remember how we said that version 2.4 was going to be the least buggy ever? Well, that might have been a bit optimistic. We had a couple of stumpers in there -- and a flurry of little ones that were probably not novel (i.e. not specific to version 2.5) but finally bubbled up to the surface. We're not going to go over the bug fixes in detail, since the release notes include a comprehensive list. Basically, those are all fixed.

Actual features!

Well, not exactly new features, but noteworthy improvements to existing tools.

- ReduceReads turns the squeeze dial up to eleven

In addition to countless bug fixes, we've made drastic improvements to ReduceReads' compression algorithm, so you can now achieve much better compression rates without compromising on the retention of informative data. Keep in mind of course that as always, you'll see much bigger gains on certain types of data sets -- the higher the coverage in your original BAM files, the bigger the savings in file size and performance of the downstream tools.

- HaplotypeCaller is faster and more accurate! No, really!

We say this every time, and every time it's true: we've made some more improvements to the HaplotypeCaller that make it faster and more accurate. Well, it's still slower than the UnifiedGenotyper, in case you were going to ask (of course you were). But on the accuracy front, we say this without reservation or caveat: HC is now just as accurate as the UG for calling SNPs, and it is in a league of its own for calling indels. If you are even remotely interested in indels you should absolutely take it out for a spin. Go. Now.

- DiagnoseTargets, all grown up

Say goodbye to the mood swings and the pimples; it looks like this tool's awkward teenager phase is finally over. We've entirely reworked how DiagnoseTargets functions so it now uses a plugin system, which we think is much more convenient. This plugin system will be explained in detail in a forthcoming documentation article.

- Functional annotation recovers some functionality

You may be aware that we had imposed a freeze of sorts on the annotation database version that could be used with the snpEff annotation. Well, we're happy to report that the author of the snpEff software package has made some significant upgrades, including a feature called GATK compatibility mode. As a result there is no longer any version constraint. We'll be updating our documentation on using snpEff with GATK soon (-ish), but in the meantime, feel free to go forth and annotate away. Just make sure to consult the snpEff manual for relevant information on using it with GATK.

Deprecation alerts

Even as the dev team giveth, the dev team taketh away.

A few annotations were removed from the VariantAnnotator stables (as listed in the release notes), mainly because they didn't work properly. With all the caveats about how GATK is research software, we're still committed to providing quality tools that do something close to what they're advertised to do, at the bare minimum. If something doesn't fulfill that requirement, it's out.

We've also disabled the auto-generation of fai/dict files for fasta references. I can hear some of you groaning all the way from here. Yes, it was convenient -- but far too buggy. Come on people, it's a one-liner using Picard. Oh, and we're no longer allowing the use of compressed (.gz) references either -- also too buggy. The space savings were simply not worth the headaches.

Version highlights for GATK version 2.4 #2259

Last updated on 2013-03-01 18:13:30

Overview

We are very proud (and more than a little relieved) to finally present version 2.4 of the GATK! It's been a long time coming, but we're certain you'll find it well worth the wait. This release is bursting at the seams with new features and improvements, as you'll read below. It is also very probably going to be our least-buggy initial release yet, thanks to the phenomenal effort that went into adding extensive automated tests to the codebase.

Important note: Keep in mind that this new release comes with a brand new license, as we announced a few weeks ago here. Be sure to at least check out the figure that explains the different packages we (and our commercial partner Appistry) offer, and get the one that is appropriate for your use of the GATK.

With that disclaimer out of the way, here are the feature highlights of version 2.4!

Better, faster, more productive

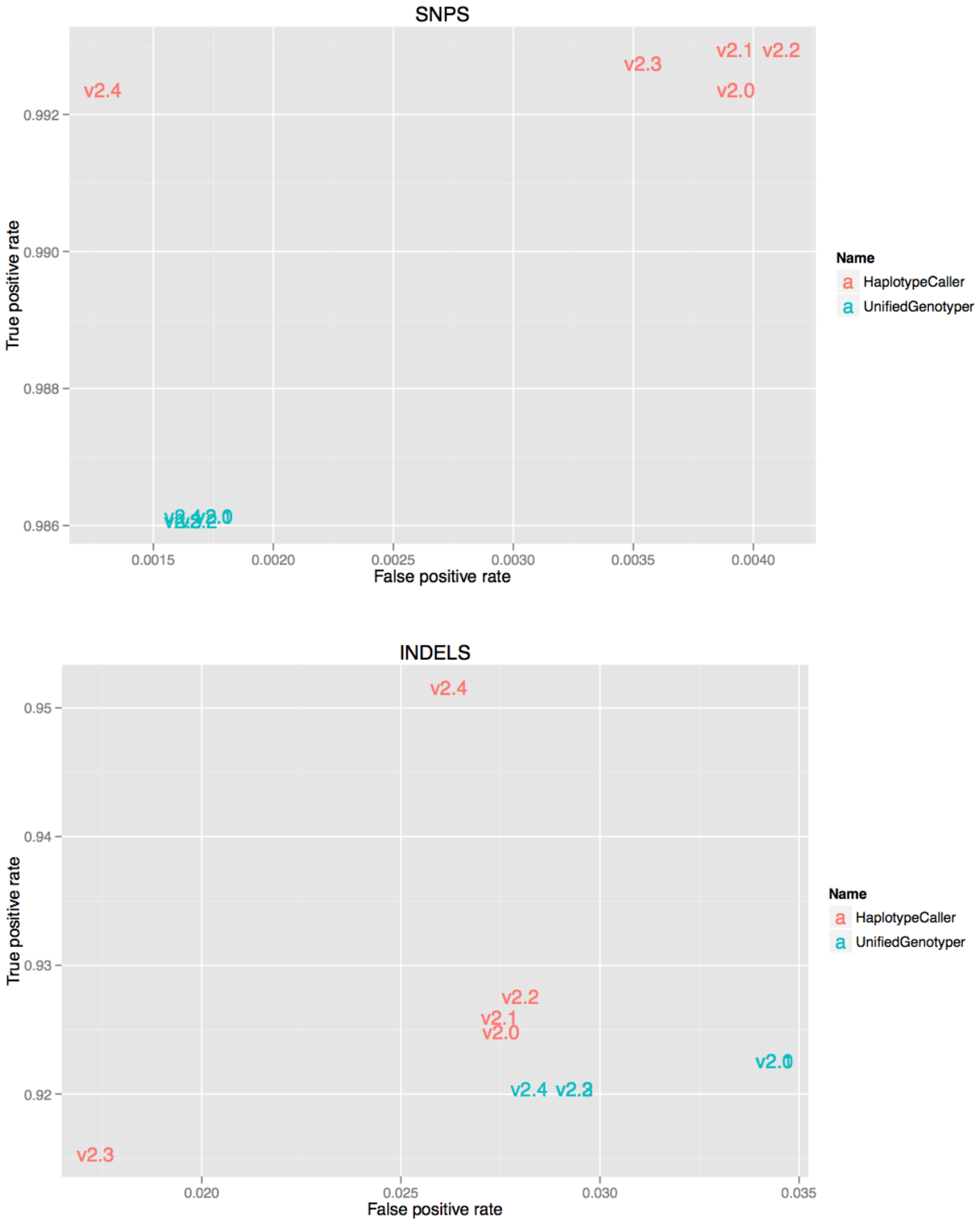
Let's start with what everyone wants to hear about: improvements in speed and accuracy. There are in fact far more improvements in accuracy than are described here, again because of the extensive test coverage we've added to the codebase. But here are the ones that we believe will have the most impact on your work.

- Base Quality Score Recalibration gets a Bayesian boost

We realized that even though BaseRecalibrator was doing a fabulous job in general, the calculation for the empirical quality of a bin (e.g. all bases at the 33rd cycle of a read) was not always accurate. Specifically, we would draw the same conclusions from bins with many or few observations -- but in the latter case that was not necessarily correct (we were seeing some Q6s get recalibrated up to Q30s, for example). We changed this behavior so that the BaseRecalibrator now calculates a proper Bayesian estimate of the empirical quality. As a result, for bins with very little data, the likelihood is dwarfed by a prior probability that tends towards the original quality; there is no effect on large bins, which were already fine. This brings noticeable improvements in the genotype likelihoods being produced from the genotypes, in particular for the heterozygous state (as expected).

- HaplotypeCaller catching up to UnifiedGenotyper on speed, gets ahead on accuracy

You may remember that in the highlights for version 2.2, we were excited to announce that the HaplotypeCaller was no longer operating on geological time scales. Well, now the HC has made another big leap forward in terms of speed -- and it is now almost as fast as the UnifiedGenotyper. If you were reluctant to move from the UG to the HC based on runtime, that shouldn't be an issue anymore! Or, if you were Â unconvinced by the merits of the new calling algorithm, Â you'll be interested to know that our internal tests show that the HaplotypeCaller is now more accurate in calling variants (SNPs as well as Indels) than the UnifiedGenotyper.



How did we make this happen? There are too many changes to list here, but one of the key modifications that

makes the HaplotypeCaller much faster (without sacrificing any accuracy!) is that we've greatly optimized how local Smith-Waterman re-assembly is applied. Previously, when the HC encountered a region where reassembly was needed, it performed SW re-assembly on the entire region, which was computationally very demanding. In the new implementation, the HC generates a "bubble" (yes, that's the actual technical term) around each individual haplotype, and applies the SW re-assembly only within that bubble. This brings down the computational challenge by orders of magnitude.

New tools, extended capabilities

We're not just fluffing up the existing tools -- we're also adding new tools to extend the capabilities of our toolkit.

- New filtering options to better control your data Â

A new Read Filter, ReassignOneMappingQualityFilter, allows you to -- well, it's in the name -- reassign one mapping quality. This is useful for example to process data output by programs like TopHat which use MAPQ = 255 to convey meaningful information. The GATK would normally ignore any reads with that mapping quality. With the new filter, you can selectively reassign that quality to something else so that those reads will get utilized, without affecting the rest of your dataset.

In addition, the recently introduced contamination filter gets upgraded with the option to apply decontamination individually per sample. Â

- Useful tool options get promoted to standalone tools

Version 2.4 includes several new tools that grew out of existing tool options. The rationale for making them standalone tools is that they represent particularly useful capabilities that merit expansion, and expanding them within their "mother tool" was simply too cumbersome.

- GenotypeConcordance graduates from being a module of VariantEval, to being its own fully-fledged tool. This comes with many bug fixes and an overhaul of how the concordance results are tabulated, which we hope will cause less confusion than it has in the past!
- RegenotypeVariants takes over -- and improves upon -- the functionality previously provided by the `--regenotype` option of SelectVariants. This tool allows you to refresh the genotype information in a VCF file after samples have been added or removed.

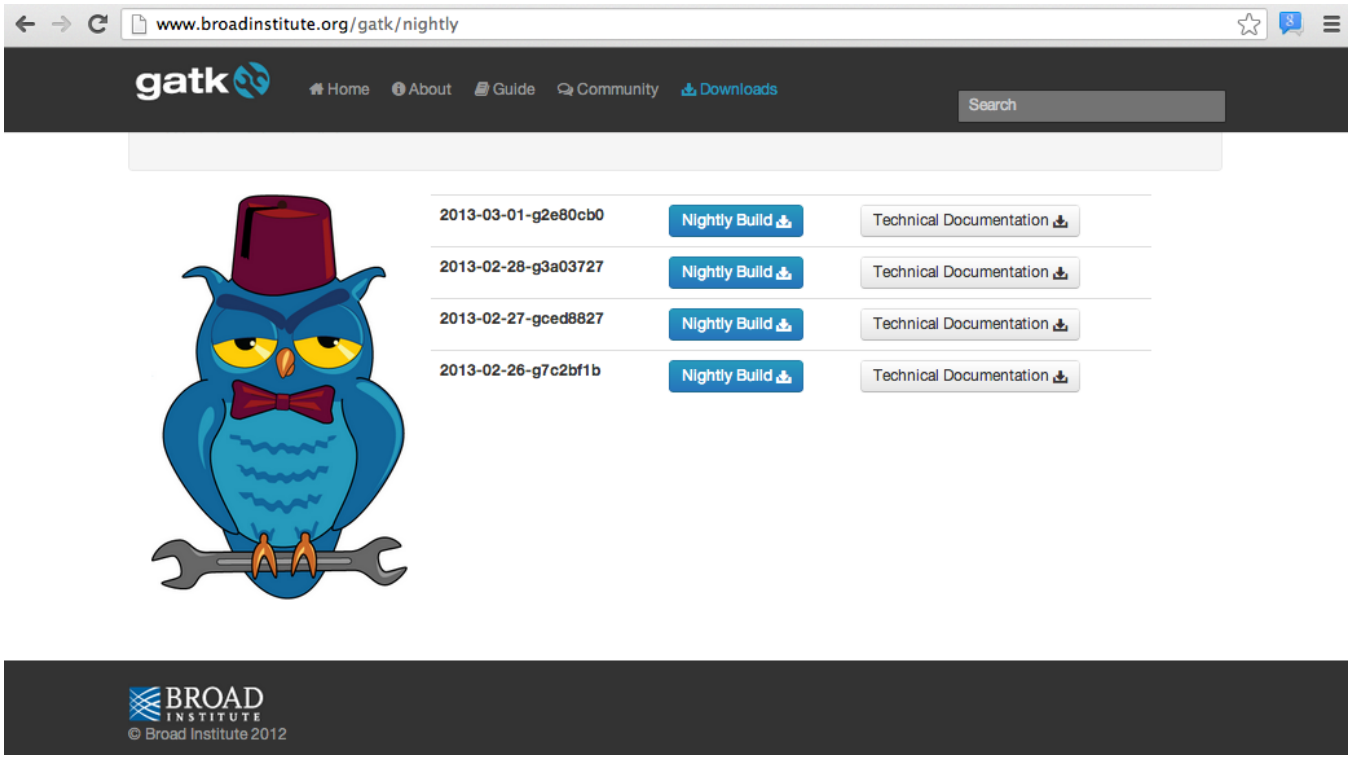
And we're also adding CatVariants, a tool to quickly combine multiple VCF files whose records are non-overlapping (e.g. as produced during scatter-gather using Queue). This should be a useful alternative to CombineVariants, which is primarily meant for more complex combination operations.

Nightly builds

Going forward, we have decided to provide nightly automated builds from our development tree. This means that you can get the very latest development version -- no need to wait weeks for bug fixes or new features anymore! **However**, this comes with a **gigantic caveat emptor**: these are bleeding-edge versions that are likely to contain bugs, and features that have never been tested in the wild. And they're automatically generated at night, so we can't even guarantee that they'll run. All we can say of any of them is that the code was able to compile -- beyond that, we're off the hook. We won't answer support questions about the new stuff. So in short: **you want to try the nightlies, you do so at your own risk**.

If any of the above scares or confuses you, no problem -- just stay well clear of the owl and you won't get bitten.

But hey, if you're feeling particularly brave or lucky, have fun :)

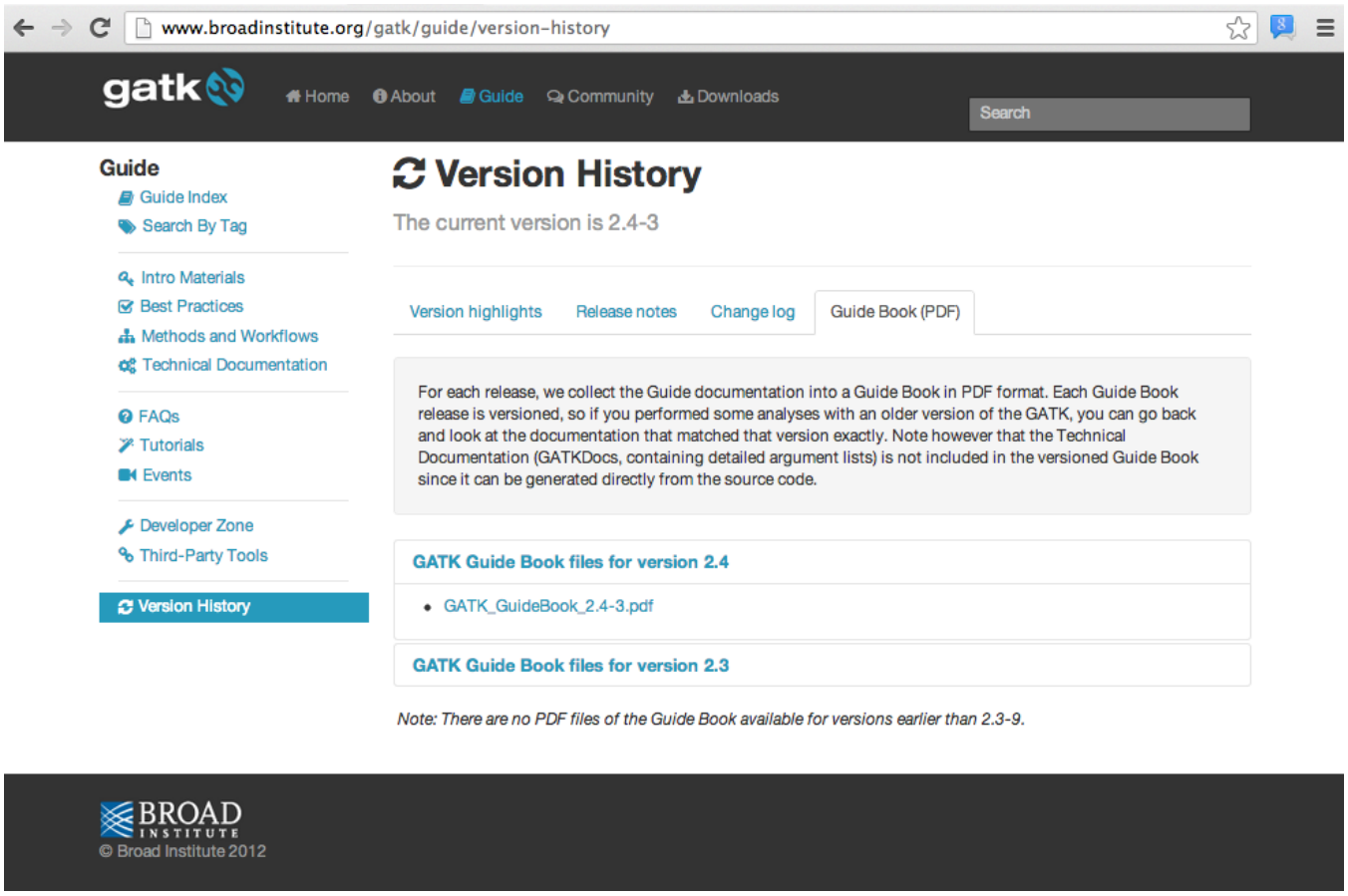


Documentation upgrades

The release of version 2.4 also coincides with some upgrades to the documentation that are significant enough to merit a brief mention.

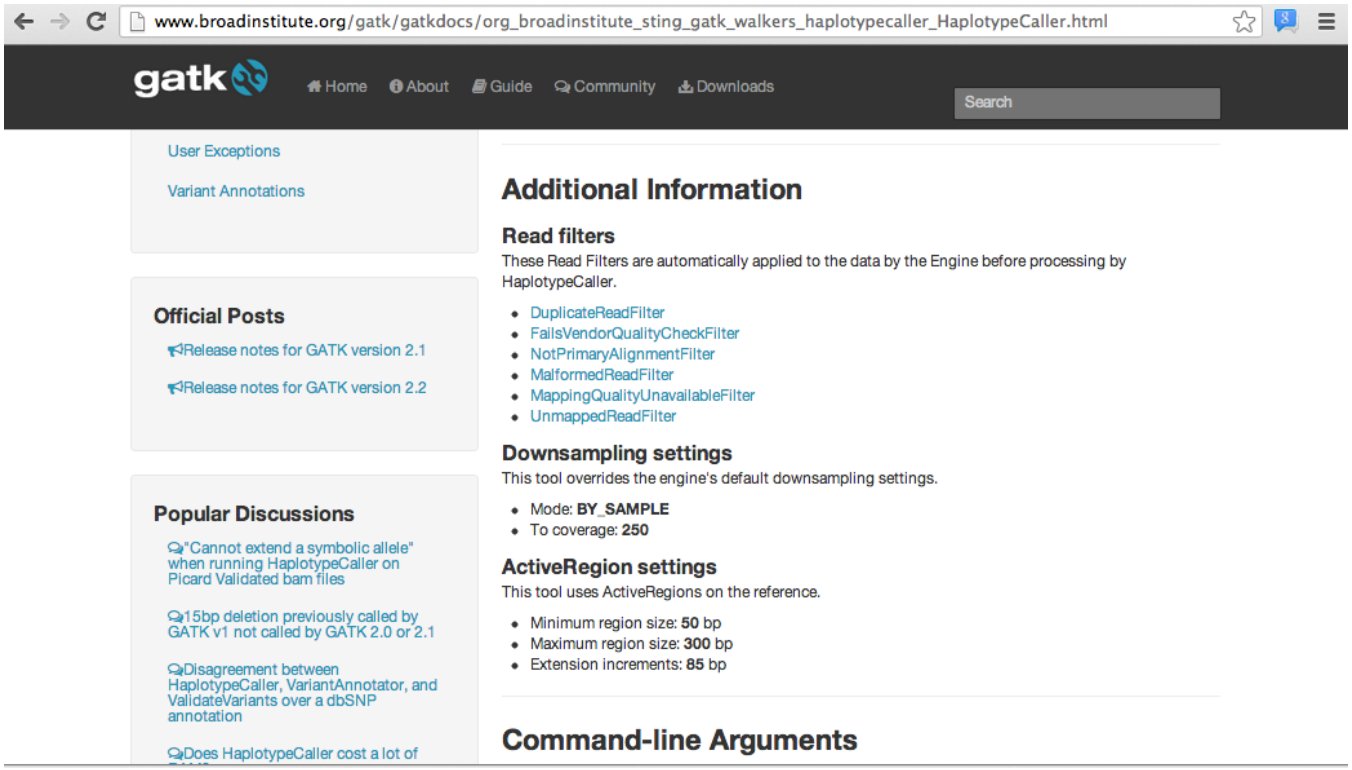
- Every release gets a versioned Guide Book PDF

From here on, every release (including minor releases, such as 2.3-9) will be accompanied by the generation of a PDF Guide Book that contains the online documentation articles as they are at that time. It will not only allow you to peruse the documentation offline, but it will also serve as versioned documentation. This way, if in the future you need to go back and examine results you obtained with an older version of the GATK, you can find easily find the documentation that was valid at that time. Note that the Technical Documentation (which contains the exhaustive lists of arguments for each tool) is not included in the Guide Book since it can be generated directly from the source code. Â



- Technical Documentation gets a Facelift

Speaking of the Technical Documentation, we are happy to announce that we've enriched those pages with additional information, including Â available parallelization options and default read filters for each tool, where applicable. We've also reorganized the main categories in the Technical Documentation index to make it easier to browse tools and find what you need.



Developer alert

Finally, a few words for developers who have previous experience with the GATK codebase. The VariantContext and related classes have been moved out of the GATK codebase and into the Picard public repository. The GATK now uses the resulting Variant.jar as an external library (currently version 1.85.1357). We've also updated the Picard and Tribble jars to version 1.84.1337.

Version highlights for GATK version 2.3 #1991

Last updated on 2013-01-24 06:00:16

Overview

Release version 2.3 is the last before the winter holidays, so we've done our best not to put in anything that will break easily. Which is not to say there's nothing important - this release contains a truckload of feature tweaks and bug fixes (see the release notes in the next tab for full list). And we do have one major new feature for you: a brand-spanking-new downsampler to replace the old one.

Feature improvement highlights

- Sanity check for mis-encoded quality scores

It has recently come to our attention that some datasets are not encoded in the standard format (Q0 == ASCII 33 according to the SAM specification, whereas Illumina encoding starts at ASCII 64). This is a problem because the GATK assumes that it can use the quality scores as they are. If they are in fact encoded using a different scale, our tools will make an incorrect estimation of the quality of your data, and your analysis results will be off.

To prevent this from happening, we've added a sanity check of the quality score encodings that will abort the program run if they are not standard. If this happens to you, you'll need to run again with the flag `--fix_misencoded_quality_scores` (`-fixMisencodedQuals`). What will happen is that the engine will simply subtract 31 from every quality score as it is read in, and proceed with the corrected values. Output files will include the correct scores where applicable.

- Overall GATK performance improvement

Good news on the performance front: we eliminated a bottleneck in the GATK engine that increased the runtime of many tools by as much as 10x, depending on the exact details of the data being fed into the GATK. The problem was caused by the internal timing code invoking expensive system timing resources far too often. Imagine you looked at your watch every two seconds -- it would take you ages to get anything done, right? Anyway, if you see your tools running unusually quickly, don't panic! This may be the reason, and it's a good thing.

- Co-reducing BAMs with ReduceReads (Full version only)

You can now co-reduce separate BAM files by passing them in with multiple `-I` or as an input list. The motivation for this is that samples that you plan to analyze together (e. g. tumor-normal pairs or related cohorts) should be reduced together, so that if a disagreement is triggered at a locus for one sample, that locus will remain unreduced in all samples. You will therefore conserve the full depth of information for later analysis of that locus.

Downsampling, overhauled

The downsampler is the component of the GATK engine that handles downsampling, i. e. the process of removing a subset of reads from a pileup. The goal of this process is to speed up execution of the desired analysis, particularly in genome regions that are covered by excessive read depth.

In this release, we have replaced the old downsampler with a brand new one that extends some options and performs much better overall.

- Downsampling to coverage for read walkers

The GATK offers two different options for downsampling:

- `--downsample_to_coverage` (`-dcov`) enables you to set the maximum amount of coverage to keep at any position
- `--downsample_to_fraction` (`-dfrac`) enables you to remove a proportional amount of the reads at any position (e. g. take out half of all the reads)

Until now, it was not possible to use the `--downsample_to_coverage` (`-dcov`) option with read walkers; you were limited to using `--downsample_to_fraction` (`-dfrac`). In the new release, you will be able to downsample to coverage for read walkers.

However, please note that the process is a little different. The normal way of downsampling to coverage (e. g. for locus walkers) involves downsampling over the entire pileup of reads in one take. Due to technical reasons, it is still not possible to do that exact process for read walkers; instead the read-walker-compatible way of doing it

involves downsampling within subsets of reads that are all aligned at the same starting position. This different mode of operation means you shouldn't use the same range of values; where you would use `-dcov 100` for a locus walker, you may need to use `-dcov 10` for a read walker. And these are general estimates - your mileage may vary depending on your dataset, so we recommend testing before applying on a large scale.

- No more downsampling bias!

One important property of the downsampling process is that it should be as random as possible to avoid introducing biases into the selection of reads that will be kept for analysis. Unfortunately our old downsampler - specifically, the part of the downsampler that performed the downsampling to coverage - suffered from some biases. The most egregious problem was that as it walked through the data, it tended to privilege more recently encountered reads and displaced "older" reads. The new downsampler no longer suffers from these biases.

- More systematic testing

The old downsampler was embedded in the engine code in a way that made it hard to test in a systematic way. So when we implemented the new downsampler, we reorganized the code to make it a standalone engine component - the equivalent of promoting it from the cubicle farm to its own corner office. This has allowed us to cover it much better with systematic tests, so we have better assessment of whether it's working properly.

- Option to revert to the old downsampler

The new downsampler is enabled by default and we are confident that it works much better than the old one. BUT as with all brand-spanking-new features, early adopters may run into unexpected rough patches. So we're providing a way to disable it and use the old one, which is still in the box for now: just add `-use_legacy_downsampler` to your command line. Obviously if you use this AND `-dcov` with a read walker, you'll get an error, since the old downsampler can't downsample to coverage for read walkers.

Version highlights for GATK version 2.2 #1730

Last updated on 2013-01-24 05:59:32

Overview:

We're very excited to present release version 2.2 to the public. As those of you who have been with us for a while know, it's been a much longer time than usual since the last minor release (v 2.1). Ah, but don't let the "minor" name fool you - this release is chock-full of major improvements that are going to make a big difference to pretty much everyone's use of the GATK. That's why it took longer to put together; we hope you'll agree it was worth the wait!

The biggest changes in this release fall in two categories: enhanced performance and improved accuracy. This is rounded out by a gaggle of bug fixes and updates to the resource bundle.

Performance enhancements

We know y'all have variants to call and papers to publish, so we've pulled out all the stops to make the GATK run faster without costing 90% of your grant in computing hardware. First, we're introducing a new multi-threading feature called **Nanoscheduler** that we've added to the GATK engine to expand your options for

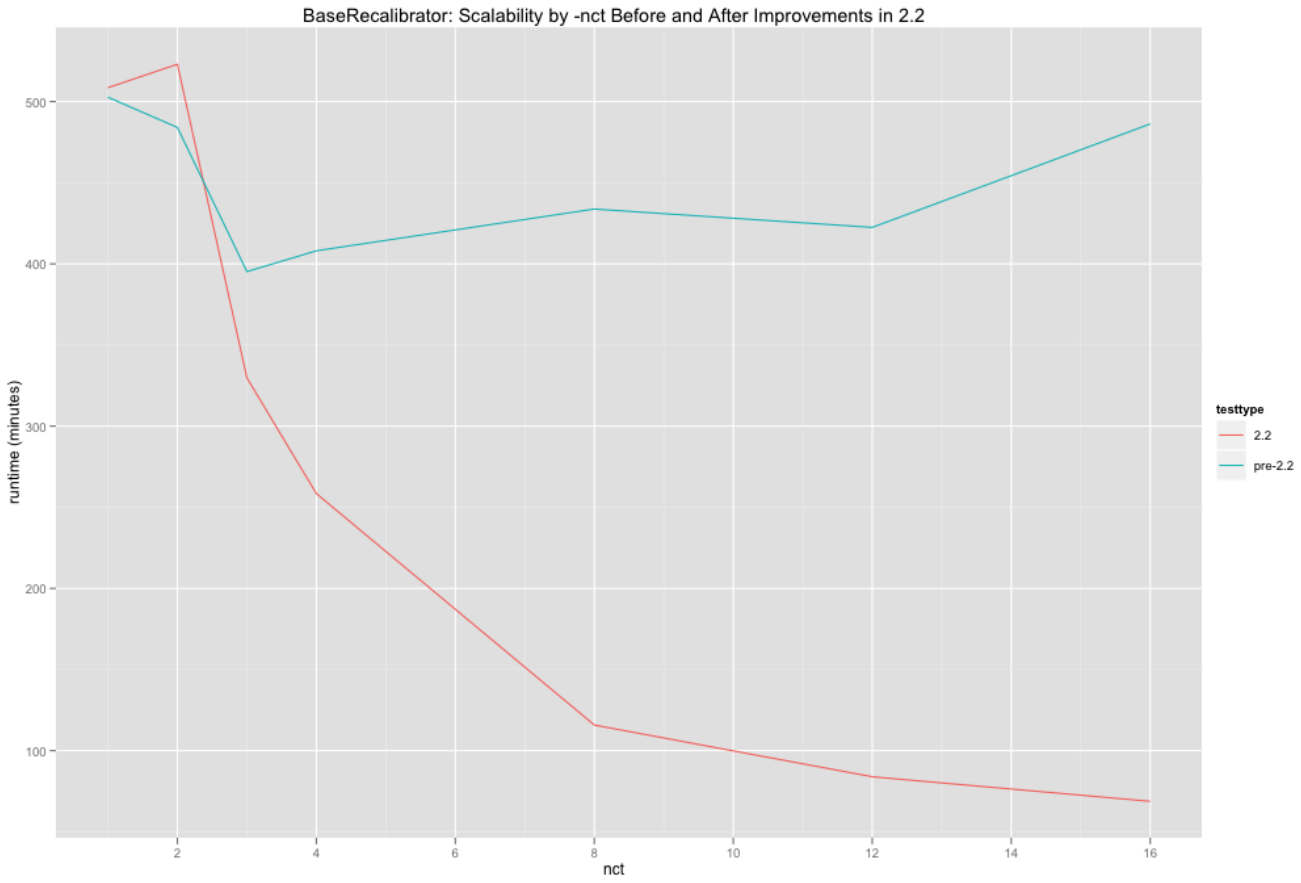
parallel processing. Thanks to the Nanoscheduler, we're finally able to bring multi-threading back to the BaseRecalibrator. We've also made some seriously hard-core algorithm optimizations to ReduceReads and the two variant callers, UnifiedGenotyper and HaplotypeCaller, that will cut your runtimes down so much you won't know what to do with all the free time. Or, you'll actually be able to get those big multisample analyses done in a reasonable amount of timeâ€¦

- Introducing the Nanoscheduler

This new multi-threading feature of the GATK engine allows you to take advantage of having multiple cores per machine, whether in your desktop computer or on your server farm. Basically, the Nanoscheduler creates clones of the GATK, assigns a subset of the job to each and runs it on a different core of the machine. Usage is similar to the `-nt` mode you may already be familiar with, except you call this one with the new `-nct` argument. Note that the Nanoscheduler currently reserves one thread for itself, which acts like a manager (it bosses the other threads around but doesn't get much work done itself) so to see any real performance gain you'll need to use at least `-nct 3`, which yields two "worker" threads. This is a limitation of the current implementation which we hope to resolve soon. See the updated document on Parallelism with the GATK (v2) (link coming soon) for more details of how the Nanoscheduler works, as well as recommendations on how to optimize parallelization for each of the main GATK tools.

- Multi-threading power returns to BaseRecalibrator

Many of you have complained that the rebooted BaseRecalibrator in GATK2 takes forever to run. Rightly so, because until now, you couldn't effectively run it in multi-threaded mode. The reason for that is fairly technical, but in essence, whenever a thread started working on a chunk of data it locked down access to the rest of the dataset, so any other threads would have to wait for it to finish working before they could begin. That's not really multi-threading, is it? No, we didn't think so either. So we rewrote the BaseRecalibrator to not do that anymore, and we gave it a much saner and effective way of handling thread safety: each thread locks down just the chunk of data it's assigned to process, not the whole dataset. The graph below shows the performance gains of the new system over the old one. Note that in practice, this is operated by the Nanoscheduler (see above); so remember, if you want to parallelize BaseRecalibrator, use `-nct`, not `-nt`, and be sure to assign three or more threads.



- Reduced runtimes for ReduceReads (Full version only)

Without going into the gory technical details, we optimized the underlying compression algorithm that powers ReduceReads, and we're seeing some very significant improvements in runtime. For a "best-case scenario" BAM file, i.e. a well-formatted BAM with no funny business, the average is about a three-fold decrease in runtime. Yes, it's three times faster! And if that doesn't impress you, you may be interested to know that for "worst-case scenario" BAM files (which are closer to what we see in the wild, so to speak, than in our climate-controlled test facility) we see orders of magnitude of difference in runtimes. That's tens to hundreds of times faster. To many of you, that will make the difference between being able to reduce reads or not. Considering how reduced BAMs can help bring down storage needs and runtimes in downstream operations as well -- it's a pretty big deal.

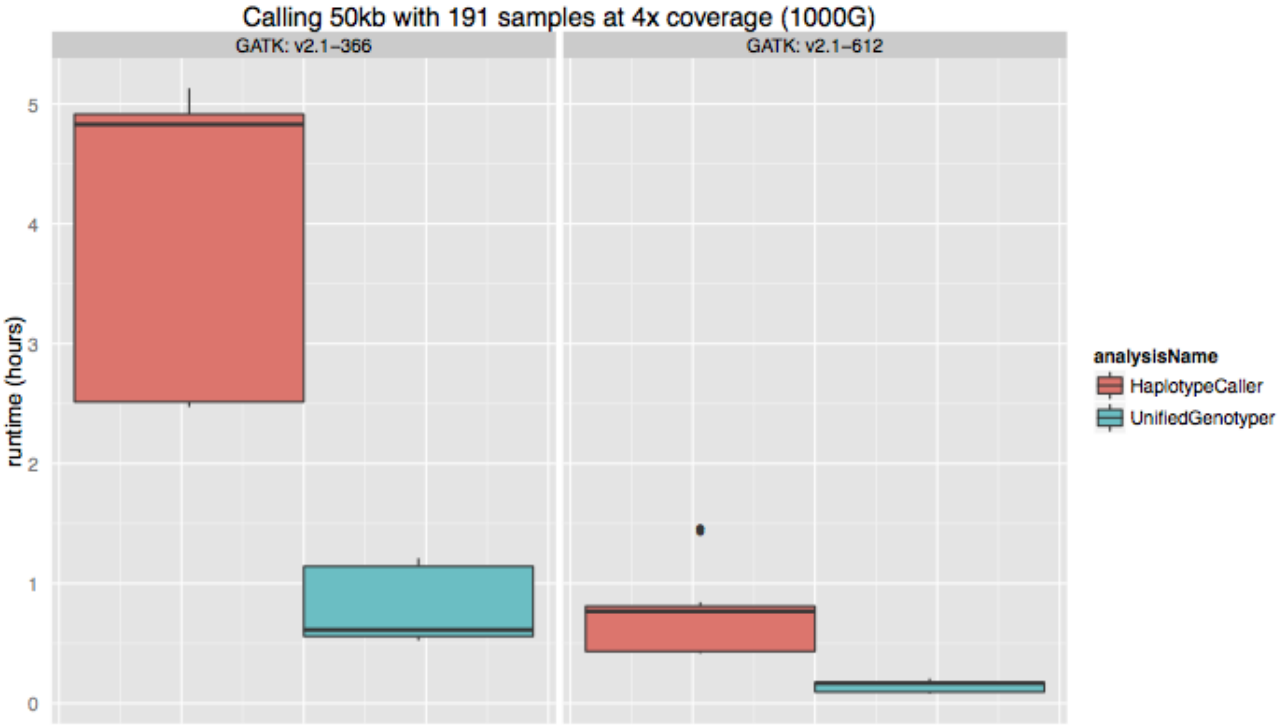
- Faster joint calling with UnifiedGenotyper

Ah, another algorithm optimization that makes things go faster. This one affects the EXACT model that underlies how the UG calls variants. We've modified it to use a new approach to multiallelic discovery, which greatly improves scalability of joint calling for multi-sample projects. Previously, the relationship between the number of possible alternate alleles and the difficulty of the calculation (which directly impacts runtime) was exponential. So you had to place strict limits on the number of alternate alleles allowed (like 3, tops) if you wanted the UG run to finish during your lifetime. With the updated model, the relationship is linear, allowing the UG to comfortably handle around 6 to 10 alternate alleles without requiring some really serious hardware to run on. This will mostly

affect projects with very diverse samples (as opposed to more monomorphic ones).

- *Making the HaplotypeCaller go* **Whoosh!** (Full version only)

The last algorithm optimization for this release, but certainly not the least (there is no least, and no parent ever has a favorite child), this one affects the likelihood model used by the HaplotypeCaller. Previously, the HaplotypeCaller's HMM required calculations to be made in logarithmic space in order to maintain precision. These log-space calculations were very costly in terms of performance, and took up to 90% of the runtime of the HaplotypeCaller. Everyone and their little sister has been complaining that it operates on a geological time scale, so we modified it to use a new approach that gets rid of the log-space calculations without sacrificing precision. Words cannot express how well that worked, so here's a graph.



This graph shows runtimes for HaplotypeCaller and UnifiedGenotyper before (left side) and after (right side) the improvements described above. Note that the version numbers refer to development versions and do not map directly to the release versions.

Accuracy improvements

Alright, going faster is great, I hear you say, but are the results any good? We're a little insulted that you asked, but we get it -- you have responsibilities, you have to make sure you get the best results humanly possible (and then some). So yes, the results are just as good with the faster tools -- and we've actually added a couple of features to make them even better than before. Specifically, the BaseRecalibrator gets a makeover that improves indel scores, and the UnifiedGenotyper gets equipped with a nifty little trick to minimize the impact of

low-grade sample contamination.

- Seeing alternate realities helps BaseRecalibrator grok indel quality scores (Full version only)

When we brought multi-threading back to the BaseRecalibrator, we also revamped how the tool evaluates each read. Previously, the BaseRecalibrator accepted the read alignment/position issued by the aligner, and made all its calculations based on that alignment. But aligners make mistakes, so we've rewritten it to also consider other possible alignments and use a probabilistic approach to make its calculations. This delocalized approach leads to improved accuracy for indel quality scores.

- Pruning allele fractions with UnifiedGenotyper to counteract sample contamination (Full version only):

In an ideal world, your samples would never get contaminated by other DNA. This is not an ideal world. Sample contamination happens more often than you'd think; usually at a low-grade level, but still enough to skew your results. To counteract this problem, we've added a contamination filter to the UnifiedGenotyper. Given an estimated level of contamination, the genotyper will downsample reads by that fraction for each allele group. By default, this number is set at 5% for high-pass data. So in other words, for each allele it detects, the genotyper throws out 5% of reads that have that allele.

We realize this may raise a few eyebrows, but trust us, it works, and it's safe. This method respects allelic proportions, so if the actual contamination is lower, your results will be unaffected, and if a significant amount of contamination is indeed present, its effect on your results will be minimized. If you see differences between results called with and without this feature, you have a contamination problem.

Note that this feature is turned **ON** by default. However it only kicks in above a certain amount of coverage, so it doesn't affect low-pass datasets.

Bug fixes

We've added a lot of systematic tests to the new tools and features that were introduced in GATK 2.0 and 2.1 (Full versions), such as ReduceReads and the HaplotypeCaller. This has enabled us to flush out a lot of the "growing pains" bugs, in addition to those that people have reported on the forum, so all that is fixed now. We realize many of you have been waiting a long time for some of these bug fixes, so we thank you for your patience and understanding. We've also fixed the few bugs that popped up in the mature tools; these are all fixed in both Full and Lite versions of course.

Details will be available in the new Change log shortly.

Resource bundle updates

Finally, we've updated the resource bundle with a variant callset that can be used as a standard for setting up your variant calling pipelines. Briefly, we generated this callset from the raw BAMs of our favorite trio (CEU Trio) according to our Best Practices (using the UnifiedGenotyper on unreduced BAMs). We additionally phased the calls using PhaseByTransmission. We've also updated the HapMap VCF.

Note that from now on, we plan to generate a new callset with each major and minor release, and the numbering of the bundle versions will follow the GATK version numbers to avoid any confusion.

Release notes for GATK version 2.7 #3136

Last updated on 2013-08-29 19:53:27

GATK 2.7 was released on August 21, 2013. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Reduce Reads

- Changed the underlying convention of having unstranded reduced reads; instead there are now at least 2 compressed reads at every position, one for each strand (forward and reverse). This allows us to maintain strand information that is useful for downstream filtering.

- Fixed bug where representative depths were arbitrarily being capped at 127 (instead of the expected 255).

- Fixed bug where insertions downstream of a variant region weren't triggering a stop to the compression.

- Fixed bug when using --cancer_mode where alignments were being emitted out of order (and causing the tool to fail).

Unified Genotyper

- Added --onlyEmitSamples argument that, when provided, instructs that caller to emit only the selected samples into the VCF (even though the calling is performed over all samples present in the provided bam files).

- FPGA support was added to the underlying HMM that is automatically used when the appropriate hardware is available on the machine.

- Added a (very) experimental argument (allSitePLs) that will have the caller emit PLs for all sites (including reference sites). Note that this does not give a fully accurate reference model because it models only SNPs. Full a proper handling of the reference model, please use the Haplotype Caller.

Haplotype Caller

- Added a still somewhat experimental PCR indel error model to the Haplotype Caller. By default this modeling is turned on and is very useful for removing false positive indel calls associated with PCR slippage around short tandem repeats (esp. homopolymers). Users have the option (with the --pcr_indel_model argument) of turning it off or making it even more aggressive (at the expense of losing some true positives too).

- Added the ability to emit accurate likelihoods for non-variant positions (i.e. what we call a "reference model" that incorporates indels as well as SNP confidences at every position). The output format can be either a record for every position or use the gVCF style recording of blocks. See the --emitRefConfidence argument for more details; note that this replaces the use of "--output_mode EMIT_ALL_SITES" in the HaplotypeCaller.

- Improvements to the internal likelihoods that are generated by the Haplotype Caller. Specifically, this tool now uses a tri-state correction like the Unified Genotyper, corrects for overlapping read pairs (from the same underlying fragment), and does not run contamination removal (allele-biased downsampling) by default.

- Several small runtime performance improvements were added (although we are still hard at work on larger improvements that will allow calling to scale to many samples; we're just not there yet).

- Fixed bug in how adapter clipping was performed (we now clip only after reverting soft-clipped bases).

- FPGA support was added to the underlying HMM that is automatically used when the appropriate hardware is available on the machine.

- Improved the "dangling tail" recovery in the assembly algorithm, which allows for higher sensitivity in calling variants at the edges of coverage (e.g. near the ends of targets in an exome).

- Added the ability to run allele-biased downsampling with different per-sample values like the Unified Genotyper (contributed by Yossi Farjoun).

Variant Annotator

- Fixed bug where only the last -comp was being annotated at a site.

Indel Realigner

- Fixed bug that arises because of secondary alignments and that was causing the tool not to update the alignment start of the mate when a read was realigned.

Phase By Transmission

- Fixed bug where multi-allelic records were being completely dropped by this tool. Now they are emitted unphased.

Variant Recalibrator

- General improvements to the Gaussian modeling, mostly centered around separating the parameters for the positive and negative training models.

- The percentBadVariants argument has been replaced with the numBad argument.

- Added mode to not emit (at all) variant records that are filtered out.

- This tool now automatically orders the annotation dimensions by their standard deviation instead of the order they were specified on the command-line in order to stabilize the training and have it produce optimal results.

- Fixed bug where the tool occasionally produced bad log10 values internally.

Miscellaneous

- General performance improvements to the VCF reading code contributed by Michael McCowan.

- Error messages are much less verbose and "scary."

- Added a LibraryReadFilter contributed by Louis Bergelson.

- Fixed the ReadBackedPileup class to represent mapping qualities as ints, not (signed) bytes.

- Added the engine-wide ability to do on-the-fly BAM file sample renaming at runtime (see the documentation for the --sample_rename_mapping_file argument for more details).

- Fixed bug in how the GATK counts filtered reads in the traversal output.

- Added a new tool called Qualify Intervals.

- Fixed major bug in the BCF encoding (the previous version was producing problematic files that were failing when trying to be read back into the GATK).

- Picard/sam/tribble/variant jars updated to version 1.96.1534.

Release notes for GATK version 2.6 #2795

Last updated on 2013-06-20 03:43:19

GATK 2.6 was released on June 20, 2013. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Important note: with this release the GATK has officially moved to using Java 7.

Reduce Reads

- Small runtime performance improvements contributed by Michael McCowan.

- Added fix for the "Removed too many insertions, header is now negative" bug.

- Fixed bug that arises in multi-sample mode and causes the tool to crash.

- Added --cancer_mode argument to force the user to explicitly enable multi-sample mode.

Unified Genotyper

- Runtime performance improvements when calling indels; calling indels in a single sample is almost 2x faster in our tests.

- Fixed bug for bad AD values in some cases.

- Fixed bug for GENOTYPE_GIVEN_ALLELES mode where it silently fails to genotype indels in some cases.

Haplotype Caller

- We have been working hard to reduce the number of false negatives (i.e. missed sites) for the Haplotype Caller and as such added a bunch of improvements to this tool. The sensitivity is now better than that of the Unified Genotyper is all of our whole genome tests for both SNPs and indels. Feel free to peruse the detailed version history for more information.

- The Haplotype Caller now annotates IDs from dbSNP properly.

- The Haplotype Caller now emits per-sample DP.

- Fixed bug for bad AD values in some cases.

- Fixed bug with error: "Only one of refStart or refStop must be < 0, not both" that arose from soft-clipped

reads at the beginning of contigs.

- Implemented a much improved version of GENOTYPE_GIVEN_ALLELES mode in the Haplotype Caller that works so much better.

Indel Realigner

- Fixed bug where secondary alignments were not being handled correctly.

Genotype Concordance

- Added an overall genotype concordance metric to the output.

- Fixed a bug in the printout of molten data in how it treated the genotypes.

Diagnose Targets

- Diagnose Targets now has an option to output missing intervals.

- Fixed bug where sometimes intervals were emitted out of order.

Base Recalibrator

- Fixed bug for reads with indel CIGAR operators (I or D) at the start/end of the read.

- Introduced a new tool, AnalyzeCovariates, to generate the BQSR quality assessment plots as a separate step, instead of doing it through the BaseRecalibrator.

Combine Variants

- We no longer add PASS to the FILTER field of unfiltered records.

Variant Annotator

- The RMSMappingQuality annotation now works properly with reduced reads.

- The various rank sum tests no longer use reduced reads in their calculations (because those reads do not represent distinct observations).

- Fixed bug in the BaseQualityRankSumTest annotation where it was not actually using the base qualities.

- Added a new annotation DepthPerSampleHC that is used by default in the HaplotypeCaller.

Miscellaneous

- James Warren contributed a patch to have references with non-suffix ".fa" parse correctly.

- We now emit the GATK version number in the header of VCFs that we produce.

- Fixed bug in the up front downsampling used by the GATK: reduced reads are no longer allowed to be eliminated during downsampling.

- dbSNP rsID matching is now smarter: variants are considered matching if they have the same reference

allele and at least 1 common alternative allele.

- We now warn users about using the GATK with RNA-seq data.

- We now check that -compress arguments are within allowable range 0-9.

- -rf ReassignMappingQuality can now be used to reassign mapping qualities to 60 before the engine filters them out with MappingQualityUnassigned.

- Fixed bug where requesting gzip VCF output with multi-threading was causing the GATK to fail.

- We now require a minimum -dcov value of 200 for Locus and ActiveRegion walkers when downsampling to coverage.

- Zero-length and repeated cigar elements are collapsed down by default in the engine.

- -ds option removed from PrintReads because it was redundant with the engine-level -dfrac argument.

- Fixed bug where the --defaultBaseQualities argument didn't always work.

- The engine now produces much more accurate read counts for Read traversals.

- Count Reads now uses a Long instead of an Integer for counts to prevent overflows.

- Locus Walkers now only try to clip adaptors when both reads of the pair are on opposite strands.

- Fixed VCF issue where PLs were capped at 32767.

- Picard/Tribble/Variant jars updated to version 1.91.1453.

Release notes for GATK version 2.5 #2572

Last updated on 2013-05-06 15:51:39

GATK 2.5 was released on April 30, 2013. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Reduce Reads

- DRASTIC improvements in the compression algorithm plus myriad bug fixes. Too many to list here; see detailed version history for more information.

Unified Genotyper

- Fixed bug for indel calling with really long reads (assigning the wrong genotypes).

- Automatic contamination fixing now works on reduced reads.

- Fixed rare bug in the general ploidy SNP likelihood model when there are no informative reads in a pileup.

- Fixed bug where haplotypes with 0 bases were being created.

- Fixed problem where our internal PairHMM was generating positive likelihoods.

Haplotype Caller

- Comprehensive performance improvements to the accuracy of calling both SNPs and indels; runtime is

also much improved (but still slower than the Unified Genotyper; we expect it to be faster than UG in the next release though). See detailed version history for more information.

- Fixed bug for calling on reduced reads (counts were not being assigned correctly).

- Fixed problem where our internal PairHMM was generating positive likelihoods.

- Can now write BAMs showing the assembled haplotypes.

Diagnose Targets

- Significantly refactored this tool; it now works with a "plugin" system (see documentation for more information).

- Fixed bug where LOW_MEDIAN_COVERAGE was output when no reads are covering the interval.

- Fixed bug where intervals were skipped when they were not covered by any reads.

Base Recalibrator

- Fixed the tool to work correctly with empty BQSR tables.

- Fixed issue where Print Reads was running out of disk space when using the -BQSR option even for small bam files.

- Fixed bug for RNA seq alignments with Ns.

Select Variants

- Fixed bug where using the --exclude_sample_file argument was giving bad results.

- Fixed bug when using the --keepOriginalAC argument which caused it to emit bad VCFs.

- Fixed bug where maxIndelSize argument wasn't getting applied to deletions.

Variant Annotator

- Added support for snpEff "GATK compatibility mode".

- Can now list available annotations by doing `java -cp GenomeAnalysisTK.jar org.broadinstitute.sting.tools.ListAnnotations`

- QualByDepth remaps QD values > 40 to a gaussian around 30.

- Removed several deprecated annotations (AverageAltAlleleLength, MappingQualityZeroFraction, and TechnologyComposition) and others are no longer marked as experimental.

Variant Filtration

- Don't allow users to specify keys and IDs that contain angle brackets or equals signs (which are not allowed in the VCF specification).

- Added feature that allows one to filter sites outside of a given mask.

Left Align Variants

- Renamed to LeftAlignAndTrimVariants.
- Added ability to trim common bases in front of indels before left-aligning.
- Added ability to split multiallelic records and then left align them.

Miscellaneous

- We removed the auto-creation of fai/dict files for fasta references because it was too buggy.
- Fixed bug where we could fail to find the intersection of unsorted/missorted interval lists.
- Fixed @PG tag uniqueness issue with BAMs we were producing.
- Fixed rare bug in GenotypeConcordance for multi-allelic sites.
- Added check for reads without stored bases (i.e. that use '*') which we do not support.
- Added support to reduce reads to CallableLoci.
- Added a new walker to split MNPs into their allelic primitives (SNPs).
- We no longer allow the use of compressed (.gz) references in the GATK.
- Picard/Tribble/Variant jars updated to version 1.90.1442.

Release notes for GATK version 2.4 #2252

Last updated on 2013-03-11 13:33:57

GATK 2.4 was released on February 26, 2013. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Important note 1 for this release: with this release comes an updated licensing structure for the GATK. Different files in our public repository are protected with different licenses, so please see the text at the top of any given file for details as to its particular license.

Important note 2 for this release: the GATK team spent a tremendous amount of time and engineering effort to add extensive tests for many of our core tools (a process that will continue into future releases). Unsurprisingly, as part of this process many small (and some not so small) bugs were uncovered during testing that we subsequently fixed. While we usually attempt to enumerate in our release notes all of the bugs fixed during a given release, that would entail quite a Herculean effort for release 2.4; so please just be aware that there were many smaller fixes that may be omitted from these notes.

Base Quality Score Recalibration

- The underlying calculation of the recalibration has been improved and generalized so that the empirical quality is now calculated through a Bayesian estimate. This radically improves the accuracy in particular for bins with small numbers of observations.
- Added many run time improvements so that this tool now runs much faster.

- Print Reads writes a header when used with the -BQSR argument.

- Added a check to make sure that BQSR is not being run on a reduced bam (which would be bad).

- The --maximum_cycle_value argument can now be specified during the Print Reads step to prevent problems when running on bams with extremely long reads.

- Fixed bug where reads with an existing BQ tag and soft-clipped bases could cause the tool to error out.

Unified Genotyper

- Fixed the QUAL calculation for monomorphic (homozygous reference) sites (the math for previous versions was not correct).

- Biased downsampling (i.e. contamination removal) values can now be specified as per-sample fractions.

- Fixed bug where biased downsampling (i.e. contamination removal) was not being performed correctly in the presence of reduced reads.

- The indel likelihoods calculation had several bugs (e.g. sometimes the log likelihoods were positive!) that manifested themselves in certain situations and these have all been fixed.

- Small run time improvements were added.

Haplotype Caller

- Extensive performance improvements were added to the Haplotype Caller. This includes run time enhancements (it is now **much** faster than previous versions) plus improvements in accuracy for both SNPs and indels. Internal assessment now shows the Haplotype Caller calling variants more accurately than the Unified Genotyper. The changes for this tool are so extensive that they cannot easily be enumerated in these notes.

Variant Annotator

- The QD annotation is now divided by the average length of the alternate allele (weighted by the allele count); this does not affect SNPs but makes the calculation for indels much more accurate.

- Fixed Fisher Strand annotation where p-values sometimes summed to slightly greater than 1.0.

- Fixed Fisher Strand annotation for indels where reduced reads were not being handled correctly.

- The Haplotype Score annotation no longer applies to indels.

- Added the Variant Type annotation (not enabled by default) to annotate the VCF record with the variant type.

- The DepthOfCoverage annotation has been renamed to Coverage.

Reduce Reads

- Several small run time improvements were added to make this tool slightly faster.

- By default this tool now uses a downsampling value of 40x per start position.

Indel Realigner

- Fixed bug where some reads with soft clipped bases were not be realigned.

Combine Variants

- Run time performance improvements added where one uses the PRIORITIZE or REQUIRE_UNIQUE options.

Select Variants

- The --regenotype functionality has been removed from SelectVariants and transferred into its own tool: RegenotypeVariants.

Variant Eval

- Removed the GenotypeConcordance evaluation module (which had many bugs) and converted it into its own tested, standalone tool (called GenotypeConcordance).

Miscellaneous

- The VariantContext and related classes have been moved out of the GATK codebase and into Picard's public repository. The GATK now uses the variant.jar as an external library.

- Added a new Read Filter to reassign just a particular mapping quality to another one (see the ReassignOneMappingQualityFilter).

- Added the Regenotype Variants tool that allows one to regenotype a VCF file (which must contain likelihoods in the PL field) after samples have been added/removed.

- Added the Genotype Concordance tool that calculates the concordance of one VCF file against another.

- Bug fix for VariantsToVCF for records where old dbSNP files had '-' as the reference base.

- The GATK now automatically converts IUPAC bases in the reference to Ns and errors out on other non-standard characters.

- Fixed bug for the DepthOfCoverage tool which was not counting deletions correctly.

- Added Cat Variants, a standalone tool to quickly combine multiple VCF files whose records are non-overlapping (e.g. as produced during scatter-gather).

- The Somatic Indel Detector has been removed from our codebase and moved to the Broad Cancer group's private repository.

- Fixed Validate Variants rsID checking which wasn't working if there were multiple IDs.

- Picard jar updated to version 1.84.1337.

- Tribble jar updated to version 1.84.1337.

- Variant jar updated to version 1.85.1357.

Release notes for GATK version 2.3 #1981

Last updated on 2012-12-18 20:21:23

GATK 2.3 was released on December 17, 2012. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Base Quality Score Recalibration

- Soft clipped bases are no longer counted in the delocalized BQSR.
- The user can now set the maximum allowable cycle with the --maximum_cycle_value argument.

Unified Genotyper

- Minor (5%) run time improvements to the Unified Genotyper.
- Fixed bug for the indel model that occurred when long reads (e.g. Sanger) in a pileup led to a read starting after the haplotype.
- Fixed bug in the exact AF calculation where log10pNonRefByAllele should really be log10pRefByAllele.

Haplotype Caller

- Fixed the performance of GENOTYPE_GIVEN_ALLELES mode, which often produced incorrect output when passed complex events.
- Fixed the interaction with the allele biased downsampling (for contamination removal) so that the removed reads are not used for downstream annotations.
- Implemented minor (5-10%) run time improvements to the Haplotype Caller.
- Fixed the logic for determining active regions, which was a bit broken when intervals were used in the system.

Variant Annotator

- The FisherStrand annotation ignores reduced reads (because they are always on the forward strand).
- Can now be run multi-threaded with -nt argument.

Reduce Reads

- Fixed bug where sometime the start position of a reduced read was less than 1.
- ReduceReads now co-reduces bams if they're passed in toghether with multiple -I.

Combine Variants

- Fixed the case where the PRIORITIZE option is used but no priority list is given.

Phase By Transmission

- Fixed bug where the AD wasn't being printed correctly in the MV output file.

Miscellaneous

- A brand new version of the per site down-sampling functionality has been implemented that works much, much better than the previous version.
- More efficient initial file seeking at the beginning of the GATK traversal.
- Fixed the compression of VCF.gz where the output was too big because of unnecessary call to flush().
- The allele biased downsampling (for contamination removal) has been rewritten to be smarter; also, it no longer aborts if there's a reduced read in the pileup.
- Added a major performance improvement to the GATK engine that stemmed from a problem with the NanoSchedule timing code.
- Added checking in the GATK for mis-encoded quality scores.
- Fixed downsampling in the ReadBackedPileup class.
- Fixed the parsing of genome locations that contain colons in the contig names (which is allowed by the spec).
- Made ID an allowable INFO field key in our VCF parsing.
- Multi-threaded VCF to BCF writing no longer produces an invalid intermediate file that fails on merging.
- Picard jar remains at version 1.67.1197.
- Tribble jar updated to version 119.

Release notes for GATK version 2.2 #1735

Last updated on 2012-11-19 13:41:24

GATK release 2.2 was released on October 31, 2012. Highlights are listed below. Read the detailed version history overview here: http://www.broadinstitute.org/gatk/guide/version-history

Base Quality Score Recalibration

- Improved the algorithm around homopolymer runs to use a "delocalized context".
- Massive performance improvements that allow these tools to run efficiently (and correctly) in multi-threaded mode.
- Fixed bug where the tool failed for reads that begin with insertions.
- Fixed bug in the scatter-gather functionality.
- Added new argument to enable emission of the .pdf output file (see --plot_pdf_file).

Unified Genotyper

- Massive runtime performance improvement for multi-allelic sites; -maxAltAlleles now defaults to 6.

- The genotyper no longer emits the Stand Bias (SB) annotation by default. Use the --computeSLOD argument to enable it.

- Added the ability to automatically down-sample out low grade contamination from the input bam files using the --contamination_fraction_to_filter argument; by default the value is set at 0.05 (5%).

- Fixed annotations (AD, FS, DP) that were miscalculated when run on a Reduce Reads processed bam.

- Fixed bug for the general ploidy model that occasionally caused it to choose the wrong allele when there are multiple possible alleles to choose from.

- Fixed bug where the inbreeding coefficient was computed at monomorphic sites.

- Fixed edge case bug where we could abort prematurely in the special case of multiple polymorphic alleles and samples with drastically different coverage.

- Fixed bug in the general ploidy model where it wasn't counting errors in insertions correctly.

- The FisherStrand annotation is now computed both with and without filtering low-qual bases (we compute both p-values and take the maximum one - i.e. least significant).

- Fixed annotations (particularly AD) for indel calls; previous versions didn't accurately bin reads into the reference or alternate sets correctly.

- Generalized ploidy model now handles reference calls correctly.

Haplotype Caller

- Massive runtime performance improvement for multi-allelic sites; -maxAltAlleles now defaults to 6.

- Massive runtime performance improvement to the HMM code which underlies the likelihood model of the HaplotypeCaller.

- Added the ability to automatically down-sample out low grade contamination from the input bam files using the --contamination_fraction_to_filter argument; by default the value is set at 0.05 (5%).

- Now requires at least 10 samples to merge variants into complex events.

Variant Annotator

- Fixed annotations for indel calls; previous versions either didn't compute the annotations at all or did so incorrectly for many of them.

Reduce Reads

- Fixed several bugs where certain reads were either dropped (fully or partially) or registered as occurring at the wrong genomic location.

- Fixed bugs where in rare cases N bases were chosen as consensus over legitimate A,C,G, or T bases.

- Significant runtime performance optimizations; the average runtime for a single exome file is now just over 2 hours.

Variant Filtration

- Fixed a bug where DP couldn't be filtered from the FORMAT field, only from the INFO field.

Variant Eval

- AlleleCount stratification now supports records with ploidy other than 2.

Combine Variants

- Fixed bug where the AD field was not handled properly. We now strip the AD field out whenever the alleles change in the combined file.
- Now outputs the first non-missing QUAL, not the maximum.

Select Variants

- Fixed bug where the AD field was not handled properly. We now strip the AD field out whenever the alleles change in the combined file.
- Removed the -number argument because it gave biased results.

Validate Variants

- Added option to selectively choose particular strict validation options.
- Fixed bug where mixed genotypes (e.g. ./1) would incorrectly fail.
- improved the error message around unused ALT alleles.

Somatic Indel Detector

- Fixed several bugs, including missing AD/DP header lines and putting annotations in correct order (Ref/Alt).

Miscellaneous

- New CPU "nano" parallelization option (-nct) added GATK-wide (see docs for more details about this cool new feature that allows parallelization even for Read Walkers).
- Fixed raw HapMap file conversion bug in VariantsToVCF.
- Added GATK-wide command line argument (-maxRuntime) to control the maximum runtime allowed for the GATK.
- Fixed bug in GenotypeAndValidate where it couldn't handle both SNPs and indels.
- Fixed bug where VariantsToTable did not handle lists and nested arrays correctly.
- Fixed bug in BCF2 writer for case where all genotypes are missing.
- Fixed bug in DiagnoseTargets when intervals with zero coverage were present.

- Fixed bug in Phase By Transmission when there are no likelihoods present.

- Fixed bug in fasta .fai generation.

- Updated and improved version of the BadCigar read filter.

- Picard jar remains at version 1.67.1197.

- Tribble jar remains at version 110.

Release notes for GATK version 2.1 #1381

Last updated on 2012-08-23 14:11:29

Base Quality Score Recalibration

- Multi-threaded support in the BaseRecalibrator tool has been temporarily suspended for performance reasons; we hope to have this fixed for the next release.

- Implemented support for SOLiD no call strategies other than throwing an exception.

- Fixed smoothing in the BQSR bins.

- Fixed plotting R script to be compatible with newer versions of R and ggplot2 library.

Unified Genotyper

- Renamed the per-sample ML allelic fractions and counts so that they don't have the same name as the per-site INFO fields, and clarified the description in the VCF header.

- UG now makes use of base insertion and base deletion quality scores if they exist in the reads (output from BaseRecalibrator).

- Changed the -maxAlleles argument to -maxAltAlleles to make it more accurate.

- In pooled mode, if haplotypes cannot be created from given alleles when genotyping indels (e.g. too close to contig boundary, etc.) then do not try to genotype.

- Added improvements to indel calling in pooled mode: we compute per-read likelihoods in reference sample to determine whether a read is informative or not.

Haplotype Caller

- Added LowQual filter to the output when appropriate.

- Added some support for calling on Reduced Reads. Note that this is still experimental and may not always work well.

- Now does a better job of capturing low frequency branches that are inside high frequency haplotypes.

- Updated VQSR to work with the MNP and symbolic variants that are coming out of the HaplotypeCaller.

- Made fixes to the likelihood based LD calculation for deciding when to combine consecutive events.

- Fixed bug where non-standard bases from the reference would cause errors.

- Better separation of arguments that are relevant to the Unified Genotyper but not the Haplotype Caller.

Reduce Reads

- Fixed bug where reads were soft-clipped beyond the limits of the contig and the tool was failing with a NoSuchElement exception.

- Fixed divide by zero bug when downsampler goes over regions where reads are all filtered out.

- Fixed a bug where downsampled reads were not being excluded from the read window, causing them to trail back and get caught by the sliding window exception.

Variant Eval

- Fixed support in the AlleleCount stratification when using the MLEAC (it is now capped by the AN).

- Fixed incorrect allele counting in IndelSummary evaluation.

Combine Variants

- Now outputs the first non-MISSING QUAL, instead of the maximum.

- Now supports multi-threaded running (with the -nt argument).

Select Variants

- Fixed behavior of the --regenotype argument to do proper selecting (without losing any of the alternate alleles).

- No longer adds the DP INFO annotation if DP wasn't used in the input VCF.

- If MLEAC or MLEAF is present in the original VCF and the number of samples decreases, remove those annotations from the output VC (since they are no longer accurate).

Miscellaneous

- Updated and improved the BadCigar read filter.

- GATK now generates a proper error when a gzipped FASTA is passed in.

- Various improvements throughout the BCF2-related code.

- Removed various parallelism bottlenecks in the GATK.

- Added support of X and = CIGAR operators to the GATK.

- Catch NumberFormatExceptions when parsing the VCF POS field.

- Fixed bug in FastaAlternateReferenceMaker when input VCF has overlapping deletions.

- Fixed AlignmentUtils bug for handling Ns in the CIGAR string.

- We now allow lower-case bases in the REF/ALT alleles of a VCF and upper-case them.

- Added support for handling complex events in ValidateVariants.

- Picard jar remains at version 1.67.1197.

- Tribble jar remains at version 110.

Release notes for GATK version 2.0 #67

Last updated on 2012-08-10 00:07:47

The GATK 2.0 release includes both the addition of brand-new (and often still experimental) tools and updates to the existing stable tools.

New Tools

- Base Recalibrator (BQSR v2), an upgrade to CountCovariates/TableRecalibration that generates base substitution, insertion, and deletion error models.

- Reduce Reads, a BAM compression algorithm that reduces file sizes by 20x-100x while preserving all information necessary for accurate SNP and indel calling. ReduceReads enables the GATK to call tens of thousands of deeply sequenced NGS samples simultaneously.

- HaplotypeCaller, a multi-sample local de novo assembly and integrated SNP, indel, and short SV caller.

- Plus powerful extensions to the Unified Genotyper to support variant calling of pooled samples, mitochondrial DNA, and non-diploid organisms. Additionally, the extended Unified Genotyper introduces a novel error modeling approach that uses a reference sample to build a site-specific error model for SNPs and indels that vastly improves calling accuracy.

Base Quality Score Recalibration

- IMPORTANT: the Count Covariates and Table Recalibration tools (which comprise BQSRv1) have been retired! Please see the BaseRecalibrator tool (BQSRv2) for running recalibration with GATK 2.0.

Unified Genotyper

- Handle exception generated when non-standard reference bases are present in the fasta.

- Bug fix for indels: when checking the limits of a read to clip, it wasn't considering reads that may already have been clipped before.

- Now emits the MLE AC and AF in the INFO field.

- Don't allow N's in insertions when discovering indels.

Phase By Transmission

- Multi-allelic sites are now correctly ignored.

- Reporting of mendelian violations is enhanced.

- Corrected TP overflow.

- Fixed bug that arose when no PLs were present.

- Added option to output the father's allele first in phased child haplotypes.

- Fixed a bug that caused the wrong phasing of child/father pairs.

Variant Eval

- Improvements to the validation report module: if eval has genotypes and comp has genotypes, then subset the genotypes of comp down to the samples being evaluated when considering TP, FP, FN, TN status.

- If present, the AlleleCount stratification uses the MLE AC by default (and otherwise drops down to use the greedy AC).

- Fixed bugs in the VariantType and IndelSize stratifications.

Variant Annotator

- FisherStrand annotation no longer hard-codes in filters for bases/reads (previously used MAPQ > 20 && QUAL > 20).

- Miscellaneous bug fixes to experimental annotations.

- Added a Clipping Rank Sum Test to detect when variants are present on reads with differential clipping.

- Fixed the ReadPos Rank Sum Test annotation so that it no longer uses the un-hardclipped start as the alignment start.

- Fixed bug in the NBaseCount annotation module.

- The new TandemRepeatAnnotator is now a standard annotation while HRun has been retired.

- Added PED support for the Inbreeding Coefficient annotation.

- Don't compute QD if there is no QUAL.

Variant Quality Score Recalibration

- The VCF index is now created automatically for the recalFile.

Variant Filtration

- Now allows you to run with type unsafe JEXL selects, which all default to false when matching.

Select Variants

- Added an option which allows the user to re-genotype through the exact AF calculation model (if PLs are present) in order to recalculate the QUAL and genotypes.

Combine Variants

- Added --mergeInfoWithMaxAC argument to keep info fields from the input with the highest AC value.

Somatic Indel Detector

- GT header line is now output.

Indel Realigner

- Automatically skips Ion reads just like it does with 454 reads.

Variants To Table

- Genotype-level fields can now be specified.

- Added the --moltenize argument to produce molten output of the data.

Depth Of Coverage

- Fixed a NullPointerException that could occur if the user requested an interval summary but never provided a -L argument.

Miscellaneous

- BCF2 support in tools that output VCFs (use the .bcf extension).

- The GATK Engine no longer automatically strips the suffix "Walker" after the end of tool names; as such, all tools whose name ended with "Walker" have been renamed without that suffix.

- Fixed bug when specifying a JEXL expression for a field that doesn't exist: we now treat the whole expression as false (whereas we were rethrowing the JEXL exception previously).

- There is now a global --interval_padding argument that specifies how many basepairs to add to each of the intervals provided with -L (on both ends).

- Removed all code associated with extended events.

- Algorithmically faster version of DiffEngine.

- Better down-sampling fixes edge case conditions that used to be handled poorly. Read Walkers can now use down-sampling.

- GQ is now emitted as an int, not a float.

- Fixed bug in the Beagle codec that was skipping the first line of the file when decoding.

- Fixed bug in the VCF writer in the case where there are no genotypes for a record but there are genotypes in the header.

- Miscellaneous fixes to the VCF headers being produced.

- Fixed up the BadCigar read filter.

- Removed the old deprecated genotyping framework revolving around the misordering of alleles.

- Extensive refactoring of the GATKReports.

- Picard jar updated to version 1.67.1197.

- Tribble jar updated to version 110.

Table of Contents

Introductory Materials

What is the GATK? 3
Using the GATK 4
High Performance 5
Which GATK package is right for you? 6

Best Practices

Best Practice Variant Detection with the GATK v4, for release 2.0 [RETIRED] 16

Methods and Workflows

A primer on parallelism with the GATK 22
Adding Genomic Annotations Using SnpEff and VariantAnnotator 29
BWA/C Bindings 38
Base Quality Score Recalibration (BQSR) 50
Best Practices for Variant Detection with GATK 51
Companion Utilities: ReorderSam 51
Companion Utilities: ReplaceReadGroups 52
Creating Amplicon Sequences 55
Creating Variant Validation Sets 57
Data Processing Pipeline [RETIRED] 61
Genotype and Validate 64
HLA Caller 71
Interface with BEAGLE Software 74
Lifting over a VCF from one reference to another 76
Local Realignment around Indels 77
Merging batched call sets 81
PacBio Data Processing Guidelines 83
Pedigree Analysis 84
Per-base alignment qualities (BAQ) in the GATK 86
Read-backed Phasing 88
ReduceReads format specifications 90
Script for sorting an input file based on a reference (SortByRef.pl) 93
Using CombineVariants 96
Using DepthOfCoverage to find out how much sequence data you have 98
Using RefSeq data 99
Using SelectVariants 102
Using Variant Annotator 105
Using Variant Filtration 105
Using VariantEval 112
Using the Somatic Indel Detector 116

Using the Unified Genotyper 121
Using the reference confidence calculation mode & generating a GVCF 124
Variant Quality Score Recalibration (VQSR) 131

FAQs

Can I use GATK on non-diploid organisms? 133
Collected FAQs about BAM files 138
Collected FAQs about VCF files 138
Collected FAQs about interval lists 139
How can I access the GSA public FTP server? 139
How can I filter my callset if I cannot use VQSR / recalibrate variants ? 141
How can I pass read filters and their arguments at the command line? 142
How can I prepare a FASTA file to use as reference? 146
How can I submit a patch to the GATK codebase? 149
How can I turn on or customize forum notifications? 149
How can I use parallelism to make GATK tools run faster? 151
How do I submit a detailed bug report? 152
How does the GATK handle these huge NGS datasets? 153
How should I interpret VCF files produced by the GATK? 157
Lane, Library, Sample and Cohort -- what do they mean and why are they important? 158
Should I use UnifiedGenotyper or HaplotypeCaller to call variants on my data? 158
What VQSR training sets / arguments should I use for my specific project? 162
What are JEXL expressions and how can I use them with the GATK? 165
What are the prerequisites for running GATK? 165
What do the VariantEval modules do? 169
What input files does the GATK accept? 173
What is "Phone Home" and how does it affect me? 177
What is GATK-Lite and how does it relate to "full" GATK 2.x? [RETIRED] 180
What is Map/Reduce and why are GATK tools called "walkers"? 181
What is a GATKReport ? 183
What should I use as known variants/sites for running tool X? 185
What's in the resource bundle and how can I get it? 186
Where can I get more information about next-generation sequencing concepts and terms? 187
Which data processing steps should I do per-lane vs. per-sample? 188
Which datasets should I use for reviewing or benchmarking purposes? 190
Why are some of the annotation values different with VariantAnnotator compared to UG or HC? 191
Why didn't the UG or HC call my SNP? I can see it right there in IGV! 192

Tutorials

(howto) Apply hard filters to a call set 197
(howto) Call variants on a diploid genome with the HaplotypeCaller 198
(howto) Call variants on a non-diploid genome with the UnifiedGenotyper 200
(howto) Compress read data with ReduceReads 201
(howto) Fix a badly formatted BAM 203

(howto) Install all software packages required to follow the GATK Best Practices. 209
(howto) Map and mark duplicates 211
(howto) Perform local realignment around indels 212
(howto) Prepare a reference for use with BWA and GATK 214
(howto) Recalibrate base quality scores = run BQSR 217
(howto) Recalibrate variant quality scores = run VQSR 224
(howto) Revert a BAM file to FastQ format 226
(howto) Run Queue for the first time 229
(howto) Run the GATK for the first time 236
(howto) Test your GATK installation 238
(howto) Test your Queue installation 242

Developer Zone

Accessing reads: AlignmentContext and ReadBackedPileup 245
Adding and updating dependencies 246
Clover coverage analysis with ant 252
Collecting output 254
Documenting walkers 255
Frequently asked questions about QScripts 258
Frequently asked questions about Scala 261
Frequently asked questions about using IntelliJ IDEA 261
GATK development process and coding standards 268
How to access the picard-public repository (containing sam-jdk, picard, tribble, and variant) 268
How to make a walker compatible with multi-threading 270
Managing user inputs 279
Managing walker data presentation and flow control 282
Output management 286
Overview of Queue 289
Packaging and redistributing walkers 291
Pipelining the GATK with Queue 296
QFunction and Command Line Options 298
Queue CommandLineFunctions 302
Queue custom job schedulers 304
Queue pipeline scripts (QScripts) 315
Queue with Grid Engine 317
Queue with IntelliJ IDEA 320
Sampling and filtering reads 322
Scala resources 323
Seeing deletion spanning reads in LocusWalkers 324
Tribble 329
Using DiffEngine to summarize differences between structured data files 332
Writing GATKdocs for your walkers 334
Writing and working with reference metadata classes 336
Writing unit / regression tests for QScripts 341
Writing unit tests for walkers 347

Writing walkers 348
Writing walkers in Scala 351

Third-Party Tools

GenomeSTRiP 352
MuTect 352
XHMM 353

Version History

Version highlights for GATK release 2.7 356
Version highlights for GATK release 2.6 359
Version highlights for GATK release 2.5 360
Version highlights for GATK version 2.4 366
Version highlights for GATK version 2.3 368
Version highlights for GATK version 2.2 373
Release notes for GATK version 2.7 375
Release notes for GATK version 2.6 377
Release notes for GATK version 2.5 379
Release notes for GATK version 2.4 381
Release notes for GATK version 2.3 383
Release notes for GATK version 2.2 386
Release notes for GATK version 2.1 387
Release notes for GATK version 2.0 390